ISSN 2278-3091
Volume 8, No.4, July – August 2019
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse30842019.pdf
https://doi.org/10.30534/ijatcse/2019/30842019

A Framework for Testing Distributed Embedded Systems

K Chaitanya[1], Dr. K Rajasekhra Rao[2], Dr. JKR Sastry[3]

[1] Scholar, Department of CSE, JNTU Hyderabad, Kilaru@gmail.com
[2]Director, Usha Rama College of Engineering, Vijayawada,
[3]Professor, Koneru Lakshmaiah Education foundation University, Vaddeswaram, AP, India

ABSTRACT

Distributed embedded systems frequently employed for implementing many of the applications such as home automation, Automobile systems, Air surveillance, and quite recently as subnets forming an IoT (Internet of things).
Distributed Embedded systems are quite complex due to the existence of heterogeneity among hardware and software and due to the existence of variance between the message flows that should happen from the protocol uses and flow requirements of the application concerned. The distributed embedded systems must be tested considering hardware, software, and the system used for networking the individual embedded systems. For undertaking testing the distributed embedded systems, many gadgets, tools, methods, and mechanisms required. Testing communication that happens among the individual embedded systems is complex. Continuous availability of the entire distributed embedded system along with the testing system is a critical requirement for undertaking comprehensive testing, which as such cannot be guaranteed. In this paper, a framework proposed for undertaking the testing of the distributed embedded system.

Key words: Distributed Embedded System, Testing embedded systems, scaffolding, Heterogeneous embedded systems, Instruction set simulator, in-circuit emulator, logic Analyzer, assert macros, Comprehensive testing

1. INTRODUCTION

Testing Distributed embedded systems comprehensively is required for developing a fail free software. An embedded system is developed using specific purpose hardware and software. Both hardware and software need to be tested, individually, and also in conjunction with each other. Testing the proper functioning of the embedded systems that are networked is also required. Testing an embedded system involves testing hardware-dependent code, independent hardware code, and testing environment required for a specific code segment.

In a distributed embedded system, several embedded systems are developed using different microcontroller-based systems which are generally heterogeneously requiring the use of middleware for doing data marshaling. Different kinds of interfaces such as RS232C, RS485, CAN, I2C, etc. are provided on the Microcontroller board, for effecting communication between the Microcontroller based systems. In a distributed embedded system network, individual embedded systems must communicate with others for implementing an application.

Each of the Microcontroller based systems is a location and as many such locations exist within a distributed embedded system. Testing must be carried at each of the locations to test the proper functioning of the local functionality and also concerning the functions running at other locations. Different kinds of methods which include scaffolding, assert macros. Instruction set simulator, logic analyzers, in-circuit emulation, etc. required for undertaking different kinds of testing such as testing for device functioning, response time, throughput, etc.,

The proper Testing environment required for undertaking testing of an embedded system. Establishing the required test environment is complex. The test environments set at each of the locations must be in working condition in conjunction with each other. Testing of embedded systems requires that the communication interfaces be working properly. Testing of embedded systems is also complex due to the existence of heterogeneity among the microcontroller-based systems used for building the individual embedded systems that act as nodes within the distributed embedded system

Scaffolding method used for testing hardware-independent code, assert macros used for testing the existence of the required environment for proper processing within the embedded system, instruction set simulator for testing the hardware in simulation mode. Logic analyzers used for

testing hardware, and in-circuit emulators, are used for testing the Target in communication with the HOST.

Different types of testing methods required for undertaking the testing of the embedded systems. Logic Analysers used for testing the hardware. Hardware independent code tested using scaffolding, assert macros used for testing hardware-independent code. Instructions set simulators are used for testing hardware-independent code and testing the devices in a simulated manner and testing the hardware-dependent code undertaken through the use of in-circuit emulators. Test cases initiated from PC transmitted to target board or the Logic analyzer for undertaking the testing and the test results are transmitted back by the Target or the Logic Analyzer back to the PC for storing and analyzing the test results.

In a distributed embedded system, both hardware and software distributed into different processing nodes connected through a network. The information communicated among the processing nodes using the network to which the nodes are connected. The kind of networking system used is the key to affect communication among the individual embedded systems. The individual embedded systems are heterogeneous that they differ in many ways, which include coding systems, parity, endian, number systems, and the interfaces. The heterogeneity among the computing nodes is also due to the existence of different interfaces requiring conversion to a specific communication standard. The issues of heterogeneity are primarily due to variances existing among different networking standards that include CAN, USB, I2C, RS485, etc. Most of the times serial, bus-based communication systems used for networking distributed embedded systems.

Use of a specific networking standard dictates the kind of testing done. The kind of testing to be done largely varies as the communication protocol changes. The interfaces available on each of the microcontroller-based system may not be suitable for communicating using a specific standard. Most of the times, there is a need to convert the native interface to the required interface so that a Microcontroller-based system participates in the network as a computing node.

Many methods presented in the literature for testing standalone embedded systems which can be investigated further to see how best these methods can be used for undertaking testing of the distributed embedded systems. The issue of setting the environment required for undertaking the testing of the distributed system needs investigation since it is one the most complex issue. Availability of the entire distributed system in working condition is another important issue that needs consideration. It is not possible to test any system unless the entire system is in working condition. Simultaneous testing at different locations in an isolated manner and an integrated manner is complex and challenging

Many processes, methods, techniques used for undertaking testing of the distributed systems in a most standard manner. A framework which encompasses all elements of testing will help to undertake to test a distributed embedded system formally. The existing methods used for a testing stand-alone embedded system must be modified, extended and included into the framework so that the framework used for undertaking the testing of distributed embedded system,

The test cases defined at system level must be broken into elementary test cases so that the elementary test cases used for undertaking testing at the individual location and the results obtained at each location are merged to arrive at over test results. One can customize the framework for suiting to the requirements of specific distributed embedded systems.

The entire distributed embedded system must be in working condition for testing a system-level test case. It is not possible to meet this kind of requirement due to the existence of many subnets within a distributed embedded network. A strategy thus is required for carrying testing without the need for an entire distributed embedded system in working condition. The strategy leads to a model which help testing carried at individual locations and the test results merged to get the overall status of the entire distributed embedded system

The system-level test cases are decomposed to elementary test cases to arrive at test cases tested at a specific location. The elementary test cases are such that they can be tested using a specific method at a specific location. The test results obtained at each location when merged will project the overall test status of the entire system. System-level test cases derived from the requirement specification of the distributed embedded system.

Many aspects considered when a distributed, embedded system tested. The aspects include interfaces, process flow, heterogeneity protocols, response time, throughput, device status, the existence of proper environment, etc. the methods required for undertaking the testing must be identified considering every element of testing carried.

A testing framework useful for testing any distributed embedded system presented in this paper. The framework, as such is extendable. To meet the testing requirements of individual distributed embedded systems

2. PROBLEM DEFINITION

Thus the problem is to create a framework that is useful for undertaking the testing of distributed embedded systems without the need to have the entire system in working condition along with the test environment system. The setting should be undertaken considering different segments of the system and the method used for testing the system.

3. RELATED WORK

In literature, many authors have presented the use of standard methods of testing either stand-alone systems or distributed embedded systems. The analysis of the methods proposed in the literature survey reveals that no specific standard methods have been in existence for testing a distributed embedded system using different methods. No comprehensive framework presented in the literature that helps in testing distributed embedded systems.

[Chen-Huan Chiang, et al., 2004] [1] Test architecture aims to transmit **JTAG** signals over a serial channel. The architecture has been developed to facilitate system testing and automatic field updating of distributed base stations situated in a wireless network. The test architectures assume that the distributed bases stations are on the same back pane and the same chassis. The architecture considers the use of boundary **SCAN** software, which is run by the processor situated within a wireless sensor node. The nodes configured by the **SCAN** software receiving instructions from a remote location. The **SCAN** software will also be able to conduct a system test and find if any system errors exist.

[Dae-Hyun Kum et al., 2006] [2] have presented a model-based system used for the development of an embedded system. The model-based systems are useful as it improves quality, and the development done is the least possible time. Simulating a system is an essay when model-based development undertaken. The system, as such, can be validated in the early stages of the development. Test cases automatically generated when systems are developed using the models. All the test cases required for validating the models and the functions can be generated using the models. Virtual prototypes o the models developed for undertaking the testing.

An electronic communication system is presented by **[Eric** Armengaud et al., 2005] [3] that connects all the individual embedded systems fitted into an automobile system. Testing of the embedded systems fitted into an automobile system required as the failure of any system may lead to disasters. Test cases are required to test the automobile system under stringent conditions. A method is presented to generate test cases based on the stimulus-response model under tough conditions. They have developed a method that is accurate and flexible, which generates test data for testing the communication within the data link layer. They have used the method for testing robustness and interoperability between the distributed systems.

Changes to the existing applications are needed due to new requirements or due to the introduction of new and sophisticated technologies. The changes made to the software may affect the code area where no changes caused, Regression testing is to be accrued to find whether the changes made to the software affected other areas of the code. Specialized hardware and software are required some times to conduct regression kind of testing. The type of testing tool selected depends on the strategy of the organization concerned. A regression testing tool is needed can be configured by the organizations as per their needs. Manual test processes are complex and time-consuming and therefore needs avoidance. Tool based testing is robust, and the process of undertaking testing rather becomes simple — [G. Walters et al., 1998] [4] have proposed an automated regression test tool that can be configured by the users as per their requirements.

[H. Thane et al., 1999] [5] have presented the testing method used for testing sequential programs by controlling the sequence of inputs fed to the application as input. Sequential test inputs are to be presented in a specific order and during specified time intervals and the time duration during which the concurrent tasks executed. One should not use sequential test techniques as they do not figure out the significance of the occurrence of the tasks.

An efficient architecture helps to develop decentralized, reliable, collaborative, and rapid applications, which need to be highly responsive real-time and distributed embedded systems. These systems have inbuilt processes for undertaking the testing. J. Russell Noseworthy, 2008] [6] have named the architecture as TENA (Testing and Training enablement architecture). A middleware built into TENA helps in code generation that is understandable through easy to understand abstractions. An excellent API included in TENA is capable of detecting programming errors at compilation time. TENA includes software components that can be used to undertake different kinds of testing.

Environment setting becomes very important for undertaking testing of any embedded system [Pei Tian et al., 2009]. [7] The basics of the environment setting must be analyzed considering the basic structure, functionalities, and characteristics of distributed software. A three-layer development pattern proposed that facilitates the setting appropriate test environment. This kind of proposal is quite difficult to implement as it is not possible to dictate a structure for the development of individual applications within distributed embedded systems.

Distributed and networked embedded systems are being used heavily in automobile, space, and many other such applications. Testing distributed applications are complex. The distributed applications are generally component-based and exhibit dynamic behavior. Dynamic interaction, structural behavior, run-time configurations, etc. makes the

testing of distributed systems complicated. It has been proved time and again that any amount of testing carried on the developed product; some unknown errors noticed during the production time. Therefore, it is necessary to undertake to test, while distributed embedded systems are in the production phase. [Peter H. Deussen et al., 2002] [8]. Therefore, there is a need to develop concepts and methods using which a distributed system tested while the system is in running mode. Online testing of distributed embedded systems is, therefore, necessary. Online testing will help to undertake the testing of functionality under limited time, resources available, complex transactions that performed between the components.

Most of the distributed embedded systems are built using fault tolerance concepts. One of the main challenges is to find the errors occurring while the distributed embedded systems are in run-state. Faults can occur at any level. The faults occurring at the PIN level normally affect the network interfaces and the communication that is built to facilitate communication between various distributed nodes which are networked. Architecture is proposed [Sara Blanc et al., 2003] [9] which consider the use of a monitor that keeps monitoring the faults occurring at the PIN level. The monitor observes the system behavior and also detects whether any failure has occurred at any of the PIN.

Interconnecting the distributed embedded systems are error-prone due to the presence of many intricate issues. Individual embedded systems as such may be error-free and become error-prone as they get connected to a network. Many methods used for undertaking the automated testing to trace out the bugs existing in the working of distributed embedded systems. [Silvie Jovalekic et al., 2008][10] have proposed cause and effect graphs which are time-dependent to describe the test cases considering the distribution and real-time properties. Tests object structure used for undertaking the testing in-depth. They have proposed a simple language describing test objects consisting of modules and connections. The language enables graphical documentation and context-sensitive protocol analysis. Symbolic representation of received messages facilitates better comprehension of system behavior.

Simulators also are used for Testing distributed embedded systems, [Steven A. Walters 1994][11] has presented a methodology for developing a simulator meant for testing a real-time distributed embedded system. The architecture deals with the various issue that includes reuse, expandability, reconfigurability, and modularity. However, the simulations model found to be inadequate for testing distributed embedded systems as the model as such is not suitable to handle inter-process communication and the requirement for proper scheduling the tasks and the need for establishing communication between concurrent tasks.

It is quite a difficult test distributed system, especially considering the issues that include synchronization, collaboration, concurrency, timing, and interoperability among the concurrent tasks. Lots of time needed for developing code required for testing a distributed embedded system. To address this issue, T. Tsai et al., 2003] [[12] have proposed a method that helps testing a distributed system quite rapidly. They have used methods for modeling test scenarios, state transitions, design, and verification patterns ripple effect analysis, regression testing, executing the test cases automatically.

Testing an embedded system can be carried by finding thin threads which represent END-TO-END testing. END-TO-END testing is an integration testing approach starting from sensing to actuating and development of a historical database. [Tsai W.T et al., 2003] [[13] have presented a method that helps to carry END-TO-END testing. They have used the concept of verification patterns used for undertaking testing. But this approach has not been applied for testing distributed embedded systems.

A massive number of individual embedded systems were tested together for reducing the time required for testing. [Yanfang Wang et al., 2010] [14] has used a master-slave system in which a PC used as a HOST for undertaking the testing. RS485 networking used for undertaking mass device testing. The arrangement used for undertaking individual device testing and not used testing of distributed embedded systems itself.

One can use a logic analyzer for testing proper working of the hardware of an embedded system by connecting probes to the junction points exposed from embedded systems. Commands are sent to a Logic analyzer so that the LA does the testing required and forward the test results back to the PC. [David E. Simon, 1999] [15] presented a method using which testing of an embedded system carried with the help of a Logic Analyzer. The testing using logic analyzers carried either is static or timing mode. The way the testing of an FPGA based board, signal integrity and memory devices using Logic analyzers has been presented by [Tektronix, 2006] [16] in their white paper,

Kyeongjoo Kim et al., [17] have presented, an analysis of streaming the data flowing across the systems presented, which viewed as the basis for proper data flow across the network. Sasi et al., [18] has presented a gaming system which used for testing an embedded system

A novel method has been presented by [J.K.R Sastry et al., 2015] [19] for networking different heterogeneous embedded systems through RS485 and bus-based serial communication system. One of the computing node connected to the network behaves like a master having full access and control of the bus.

[J.K.R Sastry et al., 2015] [20] have proposed an efficient method of networking heterogeneous systems using I2C communication system. All issues related to networking, including synchronization, timing, arbitration, design of data packets, etc. presented in this paper. [Sastry et al., 2015] have addressed the design of USB based network for connecting heterogeneous Microcontroller based system,

design of specific communication system as required by the distributed embedded application, address allocation to the salves and configuring the slaves through descriptors for making them adaptable for the implementation of distributed embedded application. The designing of the messages and controlling the flow of messages across the distributed Microcontroller based system has been presented considering a distributed embedded system that monitors and controls temperatures within a Nuclear reactor system.

Networking of a distributed embedded system is achieved through networking using the CAN-based communication system, which is a BUS based serial communication system. Every communication system requires that messages communicated in a specific sequence. The application requires the transmission of messages in a specific sequence. Both message systems must be combined to arrive at a composite communication system. [Sastry et al., 2015] have presented a novel method using which an arbitration method that takes message flows into account has been presented, leading to efficient communication using CAN-based communication system.

Protocols specify the way the messages must flow using the data packets of different types. Applications require the flow of messages in a proper sequence and order. A mapping method is requited that ensure the movement of application-specific messages while following the way a protocol dictates the flow of messages.[Sastry et al., 2017] [23] have proposed a method of organizing the movement of application-specific messages without

[K. Chaitanya et al., 2018] [24] have presented the way testing of a distributed system carried when networked using CAN protocol and using the scaffolding method for testing. They have also presented a method [25] of testing a distributed embedded system using networked through an RS485 network and using the scaffolding method for undertaking testing of the distributed embedded system.

[Chaitanya et al., 2017] [26] have proposed a method using which testing of a distributed system can be undertaken using assert macros to find the existence of the required environment for undertaking specific embedded processing. Assert Macros are inserted into the code dynamically either through the established pointers or through an interactive process. Macros are generated based on test scripts, and the same inserted into the embedded application as in-line code. Instruction set simulators used by [Chaitanya et al., 2017][27] for testing embedded systems, especially testing for throughput and response time while simulating the hardware devices which meant for carrying Input/output. [Chaitanya et al., 2017][28] have used instruction set emulator for undertaking the testing the functioning of the Hardware and software considering the Target with the test cases initiated from HOST. Logical analyzers have been used by [Chaitanya et al., 2018] [29] for testing the proper functioning of the hardware. The testing of the proper functioning of the hardware is undertaken by Logic analyzers, which fed with the test cases initiated from HOST.

[K. Chaitanya et al., 2018] [30] have presented the way testing of a distributed system carried through the different testing method by using a repository of master test cases and integrating them. [Chaitanya, 2013] [31] have explained the complications of Embedded Systems that can occur in Agriculture Technology by using a Customized Software.

4 COMPARATIVE ANALYSIS OF THE RESEARCH FINDINGS

A Comparative sample analysis of the research carried presented in Table 1. From the table, one can see that no method presented in the literature is suitable for undertaking testing of the functioning of the entire system. The methods presented as such address some part of the testing of the distributed system. None have covered the issue of testing the flow of proper order.

5.PROTOTYPE DESCRIPTION - DISTRIBUTED EMBEDDED SYSTEM

A distributed embedded system developed through 5 embedded systems which are heterogeneous with different native support for communication. The Microcontroller based systems used for establishing a distributed embedded system include 89c51, AT89S52, PIC18F4550, ATmega328, and LPC2148, used as a central server/master. These embedded systems are interconnected using either of the communication protocols that include CAN, I2C, USB, and RS485 using the native communication ports or through a port, conversion using hardware converters.

89c51 Microcontroller used for sensing temperature-1 and transmitting it to the central server, which is LPC2148. AT89S52 used for sensing temperature-2 and transmitting it to the central server, which is LPC2148. The central server interfaced with a PC for feeding the reference temperatures and also to maintain a database of sensed data. The operation of PUMP-1 controlled through PIC18F4550 and ATmega328 used for controlling the operation of PUMP-2. The central server sends message to PIC18F4550 for controlling the running of the PUMP-1 so is the case with PUMP-2, the controller of which is ATmega328 is sent a message for controlling the operation of the PUMP-2.

The Embedded system that acts like a master coded with a component that computes the temperature gradients and asserts the buzzer or otherwise based on the absolute difference of the temperatures. The master coded with all the communication components for communication with the rest of the embedded systems connected on the same network.

6. INVESTIGATIONS AND FINDINGS

6.1 Synchronizing application-specific messages with protocol-specific messages

One of the main issues addressed is related to message flow required for implementing the requirements of an application. For every application, messages must flow according to the design to meet the functional requirements.

6.1.1 Architectural design for implementing message flow system

A distributed embedded system realized through interconnecting a set of embedded systems using one of the bus-based serial communication systems that include CAN, I2C, RS485, USB, etc. Figure 1 shows a typical networking diagram using any of the communication systems. Every embedded system is connected to the BUS directly using its native port or through trough a converter that converts the native interface to the actual bus-related interface.

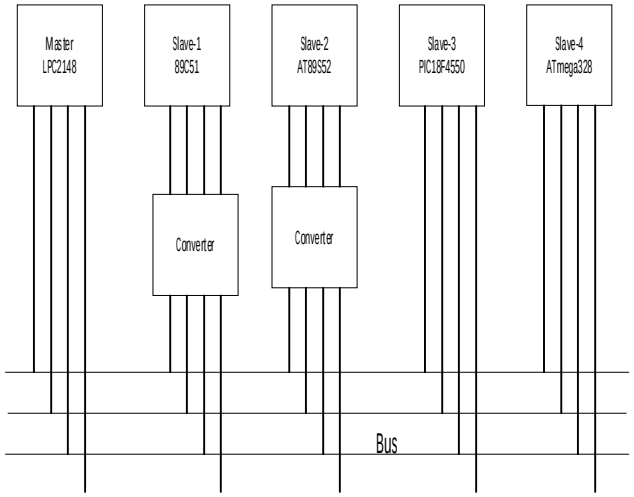

Figure 1: Typical distributed embedded system

Every embedded system will have firmware through which transmission or reception of the data undertaken by implementing specific communication software. Communication is undertaken using this communication software. The typical architecture followed for implementing the communication system within a distributed embedded system is shown in Figure 2 — the heterogeneous issues handled within the software components implemented on the transmitting side.

The communication should take place as defined in the protocols. Messages must flow in a specific sequence while at the same time following the sequence of message flow as defined in a specific protocol. The flow of messages, however, vary greatly varies from protocol to protocol. The architectural diagrams are showing the software components and the way the component interacts.

Master initiates the communication by the master by using RTR (Remote transmission request) for want of Temperature-1 and Temperature-2 which are transmitted by 89C51 and AT89S52 in that sequence and for transmitting a pump control message to PIC18F4550, and ATmega328.

6.1.2 Addressing the devices on the Network

For effective communication, every device that participates in the network addressed so that the master and the device will be made known of the address assigned. The addresses related to each of the device stored in the internal registers. Addresses assigned to the devices in many ways

1. Statically setting the addresses to the devices through toggle switches and programing the same within the master
2. Each device manufactured with a specific address and the address is transmitted to the master by the device when the device interfaced with the bus.
3. Master assigns the address to the slave at the time the device interfaced with the bus

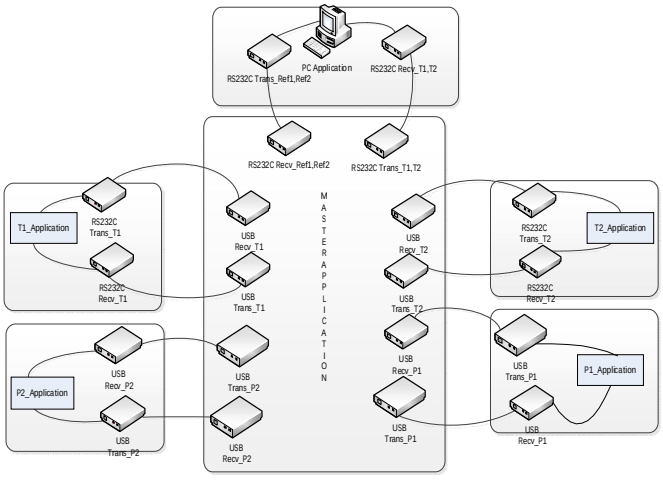

Figure 2: System Architecture for effecting communication among distributed embedded systems

The addresses of the devices stored in the master by following one of the methods mentioned above so that the master will be able to address the devices.

6.1.3 Arbitrating the bus

The BUS is always under the control of the master so that the master can initiate any communication. A slave must get hold of the bus for responding to the master. Some of the initiatives include the transmission of the device address

along with the message on the bus. Every slave will read the bus, but only the device that has the address will respond. The second method is arbitration. The devices will arbitrate using its arbitration string to gain control of the bus before transmitting. Master will always be the recipient of the response from the slaves. An arbitration string is required to be attached to each of the devices.

6.2 Message flow and priority setting

6.2.1 Message Flow Sequence

Messages must flow in a sequence as per an application requirement; for example, the master must request for temp-1 input before the embedded system responsible for sensing Temp-1 can transmit. A typical message flow sequence, as shown in Table 2.

Table 2: Message flow sequence

Message sequence Number	Message Description
1.	Master to request for Temp-1
2.	ES-1 to send temp-1
3.	Master to send the request to PUMP-1 for operational control of the PUMP-1
4.	PUMP-1 to send the pump status to the master
5.	Master to request for Temp-2
6.	ES-2 to send temp-2
7.	Master to send a request to PUMP-2 for operational control of the PUMP-2
8.	PUMP-2 to send the pump status to the master
9.	Master to activate the buzzer as per the temp-1 and temp-2 difference

These messages generated through different tasks run concurrently within the master system. Therefore, there is a need to sequence the messages even though the messages are generated randomly by concurrent tasks running within the master. Besides, the number of packets is to be transmitted based on the protocol used for affecting the messages to be transmitted from the master side — priorities assigned to the messages. Messages communicated as per priorities pre-assigned to achieve transmission of the messages in the proper sequence.

6.2.2 Setting dynamic allocation table

A dynamic Table 3 built within the memory of the master system and the processes running within the master use the master table for effecting the communication.

6.2.3 Affecting the message Flow system

Generally, the master has control of the bus. It is the master which initiates the communication by releasing the request message, and the related salve must be able to access the bus and respond to the master. There must be protocol implemented to access the bus. Slaves have to arbitrate to gain access to the bus. The application-specific messages prioritized such that messages flow as per the protocol sequence. The message which flow on the bus is attached with the priorities and written to a queue as the messages generated concurrently. The messages are pulled from the queue and sent on the bus. The arbitration strings of the slaves set according to the sequences in which the slaves must receive the messages. Figure 3 shows the working priority-based message system that implements the application-specific message flow system implemented for distributed embedded system.

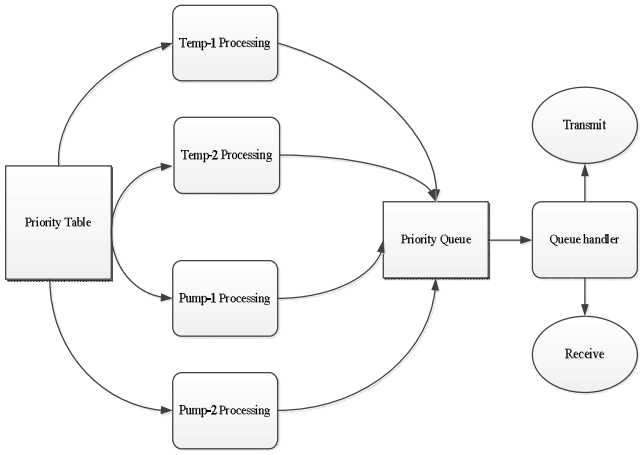

Figure 3 : Priority-based message dispatching method.

6.3 Testing Framework

The functional requirements of the distributed system traced first. To ensure functional requirements are met fully by the distributed embedded system, testing requirements have to be specified. The testing requirements as such achieved through undertaking appropriate tests considering the entire distributed system as a whole. Carrying testing across all the embedded systems connected in a network as such is quite complicated and most of the times become infeasible. A framework as such required for undertaking testing of a distributed embedded system using an analogy.

Different kinds of methods required for undertaking testing of a test case and also may lead to undertaking testing at different locations. Each testing requirements can be broken down to sub-test cases such that sub-test cases tested at a specified location. The results obtained out of testing different sub-test cases can be merged to find the testing outcome obtained when testing is done considering the entire distributed embedded system as a whole.

A single test method would be sufficient for undertaking testing using a sub-test, and that too needs to be carried at a single location. This framework captured in Table 4.A single test case tested across the distributed system can be broken into multiple sub-test cases, each one tested at a single location. A test number is attached to each sub-test case, which is the extended number of its master test case number. The sub-test cases executed at a suitable location. The test results achieved through testing of sub-test cases are merged to get the overall test results obtained across the entire distributed embedded system.

6.3.1 Setting Environment for testing distributed embedded systems

For undertaking testing at any of the location for undertaking any test using a specific method, the testing environment tested. The process used for setting the environment required shown in Figure 4.

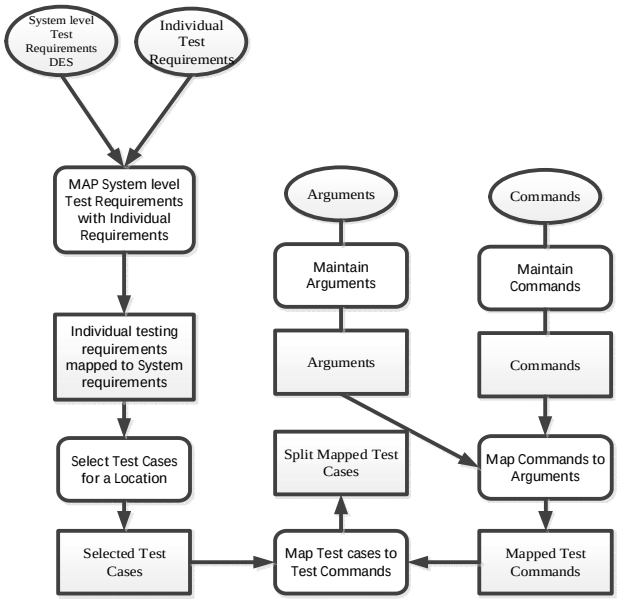

Figure 4: Environment setup for undertaking testing using Scaffolding method

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 used for undertaking testing of a Distributed embedded system can be pre-identified based on the functional requirements of a distributed embedded system. These test cases split into sub-test cases tested at a specific location. One can also set a method used for testing at each location can also be selected by the tester. The tester should have full knowledge of the methods to be used for undertaking a specific type of test. One can determine a test case for undertaking a test at a location using a specific method, test data, however, needs to be generated.

Repositories of commands maintained that indicates a kind of testing conducted at a location. Each of the test cases can be associated with a command, and the commands are associated with certain arguments. Some of the commands may involve many locations, especially when the testing of communication between the distributed elements undertaken. Commands can define with an argument that is a kind of test input passed for undertaking the testing. Table 5 shows some of the commands and their related command-line arguments. Command-line repositories are developed and maintained as one of the initial steps to be undertaken as a part of an environment setting.

Test commands are associated with the test cases numbered for remembrance and recall. This kind of mapping of test serials to the commands in the command repository provides a simple environment required for undertaking testing at different locations. A sample elementary test cases decomposed from master test cases. Details of a sample test environment shown in Table 6.

When a command is issued, or a test case through test script is issued functions of the firmware must be called in a sequence so that testing carried. The test data stored in memory variables functions called and the output stored in the memory variable reported as test results. The functions contained in the code traced, and a repository of the functions created, as shown in Table 7. The mapping of the test script to the function calling sequence shown in Table. 8. Using these tables testing is carried by calling the functions in the sequence required

6.3.2 Testing through Scaffolding

80% of the ES code is independent hardware code and therefore can be tested on HOST, the computer on which ES code is developed, cross-compiled, linked and code relocated. The remaining code is Hardware dependent code. The hardware-dependent code could be scaffolded to simulate the behavior of the hardware, thereby making it be independent hardware code. The entire code thus can be made to be independent hardware code so that the code compiled, linked, and executed on the HOST. A separate software component added to the ES application that will facilitate the testing of the code using the test scripts generated by following a different process.

Interrupt service routine called when a device interrupts the CPU for handling the I/O. Since Scaffolding software comments all the code related to the device, Interrupt routines called by the scaffolding software for simulating the behavior of the device. Similarly, time management is done through the timer initiating the interrupt to CPU. Since the functioning of the timer is to be scaffolded, the scaffolding

software must call the timer function on its own when time management is required. When a certain type of testing undertaken, the related functions connected with the test case must be called in a specific sequence so that the required testing carried.

Testing using the scaffolding method is complicated, especially when the test results generated by scaffolding software fed as a test case for some other scaffolding software run on a different PC. In this case, communication established between two units of scaffolding software run on different HOST based machines. Scaffolding generally employed for undertaking event-based and time-based testing and when the unit and integration testing is needed especially when the functionality distributed among several embedded systems connected on the same network. The process involved in undertaking testing through scaffolding software shown in figure 5. The hardware-dependent code is made hardware-independent either by simulating the Hardware or simply generating the inputs to be received from HW or outputs to be directed to HW by randomly generating them for either inputting or outputting.

Figure 5 shows the process involved in undertaking testing at a Location using scaffolding method. Test scripts generated for undertaking Testing at a distributed location. The test scripts are generated using the mapping of test cases with the commands mapped with different types of arguments. The generated test cases stored to a disk as OS file called Test Script file. The following algorithm used for generating test scripts

Algorithm for test script generation:

Do the following process for each test case to which the commands mapped.

a. Select a Test case
b. Select 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 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. Test commands are mapped to test cases using which test scripts generated. Code elements are mapped to commands to indicate the way code executed. Original code is read through to include a parser into it and comment all those code lines that are related to Hardware and the code replaced with the code that assigns some test data into the registers that are related to hardware devices. The parser reads the test scripts, interpret the same, find the functions and the calling sequence, and call the functions in the sequence required by passing on the test data. Each script generated on a common syntax. The string built with a command followed by its arguments; Table 9 shows the scripts that are generated by the Script generator

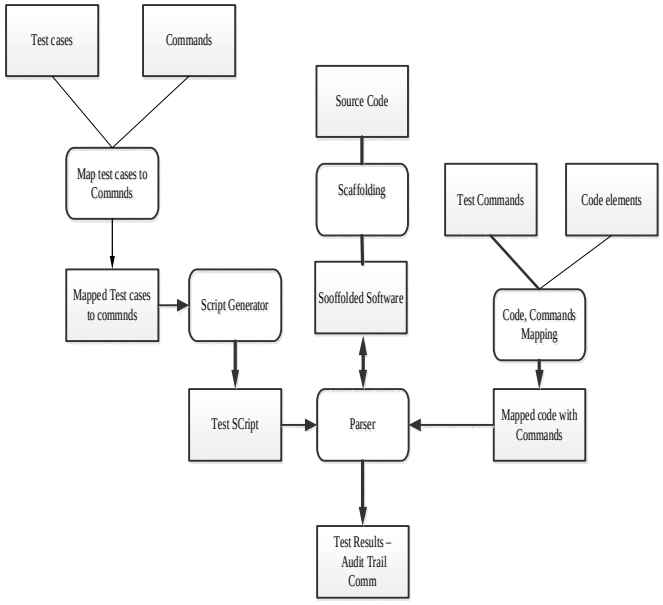

Figure 5. Process flow for testing at a Location using the method scaffolding

Undertaking testing through a parser

A parser added to the scaffolded code which is capable of reading the script file. A parser reads a test script from the script file, parses the command and its related arguments. Parser fetches the list of functions and the sequence of functions to be executed. The parsers create pointers to the function and enter the functions in a priority queue. Another process fetches the pointer from the queue and keeps executing the functions and writes the output along with the test case to the audit trail. The working of parsing and queue processing shown in figure 6.

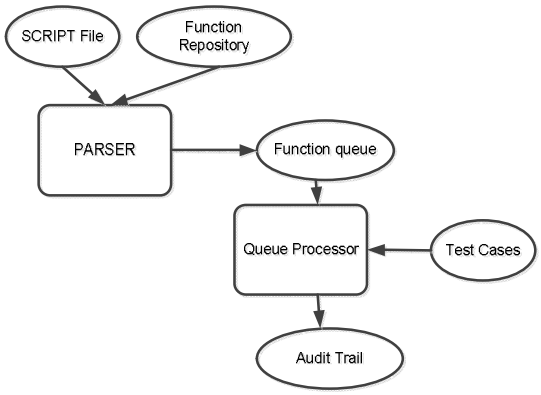

Figure 6: Testing embedded systems through parsing and queue processing

The ES application is a set of functions which called in a sequence at any of the distributed locations for undertaking testing of a particular test case. Testing is carried by finding the functions and the sequence in which the functions

executed and then calling the functions in that sequence. The arguments assigned to the respective memory variables and then the functions called in a specific sequence. The test result stored in a memory variable reported as a test result. The test results generated through scaffolding shown in Table 10.

6.3.3 Testing through Assert macros

An Assert macro is one good technique to test the existence of a particular Hardware environment required for executing some software segments. If the required hardware environment is not in existence, the software 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 done in a 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 FAILS 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 the status of radio, whether any of the control variables set to NULL, the existence of a frame or otherwise, etc. Programming languages support different kinds of assertions to verify the existence of a proper environment.

In the cases of the embedded system, a proper environment must exist for a piece of code executed properly. It is better to know about the bugs before the same happens. Assert macros will report if any error exists before a Piece of application code is executed. Testing of code done on the HOST with the assert macros inserted at strategic points where proper existence of the environment is to verified. When an assert macro returns an error, the program execution suspended as proper execution of the code is not possible after the error occurs. If no error found, the code execution continues as usual. The execution of the assert macros controlled through compilation of the code with NDEBUG option. The assert macros should not crash the system while in execution or should not degrade the performance of the system. Assert macros must be simple and very small in code size. Assert macros server many purposes which include documentation when commented, undertake to test of the existence of a suitable environment and to make debugging easier,

Figure 7 shows the process involved in undertaking testing at a particular location using assert macros. Test macros are generated using the details stored in Table 11. A separate process merges into the source code by way of recognizing 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.

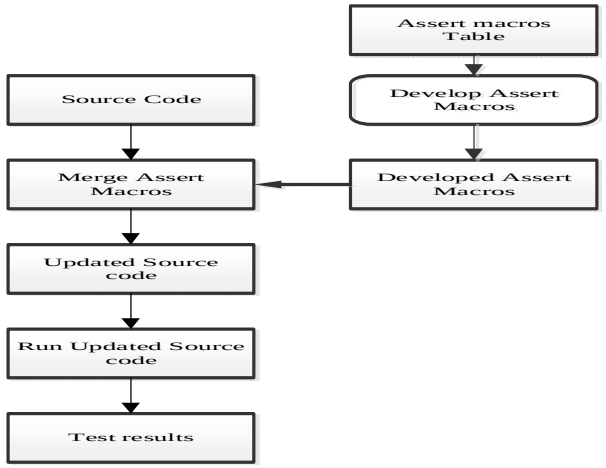

Figure 7: Process flow for undertaking testing using Assert macros

Generation of assert macros

Test macros generated considering parameters used and the kind of testing that must be undertaken using the assert macros. Table 11 shows the generated assert macros.

Testing is done using the test macros inserted into the source code and then running the program after compilation and building the object code. The test results obtained shown in Table 12**.**

6.3.4 Testing through Instruction set Simulator

Firmware always developed on the host as the target will not have any development support. The code developed on the HOST initially tested on the HOST, and then the tested code is moved to the target for storing in RAM or ROM whichever is the case. Firmware as such recognized in terms of two components which include hardware-independent code and hardware-dependent code.

Instruction set simulators will be handy to test the response time, throughput, and portability issues. A simulator is a software that runs on the host and simulates the behavior of Micro Processor and the memory on the target machine. The simulator has the knowledge of Locator output, architecture and instruction set of the target Micro Processor

Simulators used for testing using memory for registers, program counters, and address registers and data buffers. The instructions are read from memory and converted to instructions equivalent to the target machine. Simulators also support a Macro Language using which testing scenarios submitted as input to the simulators. The simulator can report response time in terms of the number of Target Machine instructions executed, the count of instructions executed, or the number of bus cycles used and the average response time computed by multiplying with average instruction execution time. Simulators can also execute the start-up code and interrupt service routines written in assembly language. Simulators also help in testing the built-in peripherals such

as a timer, DMA, UART, etc. as the simulation of such built-in simulators is quite possible. Simulators have 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 investigated whenever the errors traced while running the application on the HOST. The test cases 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 second stage using which the Audit Trail conducted.

Test cases presented as commands to instruction set simulator and the arguments that are associated with the commands assigned with range values. The process flow for undertaking testing using instruction set simulator shown in Figure 8. Test scripts generated for undertaking Testing at distributed locations for undertaking comprehensive testing of a distributed embedded system. The test scripts are generated using the mapping of test cases with the commands mapped with different types of arguments. The generated test cases stored in an OS file called Test Script file. The following algorithm used for generating test scripts.

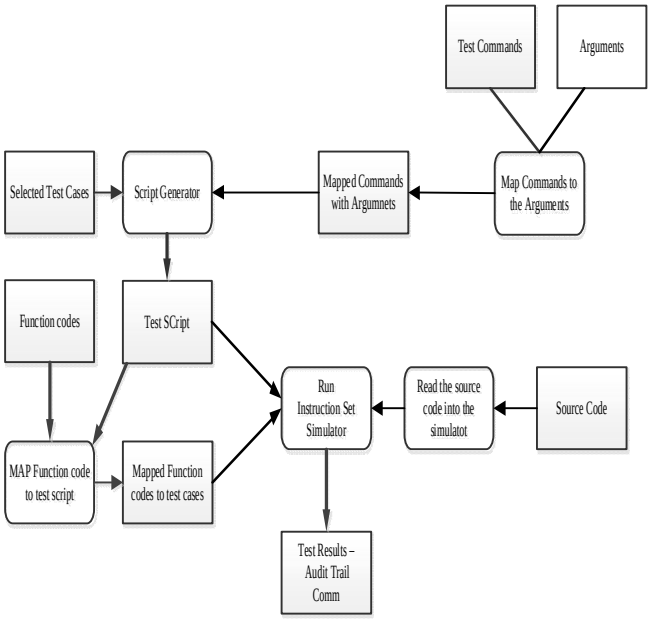

Figure 8: Process flow for undertaking testing at an Individual Location

Algorithm

Step-1

Read the source code into the simulator

Step-2

Reading a test case into the simulator
Do the following process for each test case to which the commands mapped.

a. Select a Test case for testing through an instruction set simulator
b. Select Script commands related the test cases
c. Select the arguments related to the command
    i. For each of the argument, select argument type and value range 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 simulator

Step-3

Run the instruction set simulator and record the test results into a test results file

The test scripts generated shown in Table 13

Testing carried at each of the locations using the process mentioned above flows using the testing cases that converted into test scripts. The test results obtained shown in Table 14.

6.3.5 Testing through In-circuit Emulator

In-circuit emulators help in testing hardware-software embedded into the target through test instructions /commands passed through the HOST. The emulators help to undertake to test for throughput, response, portability, etc. undertaken within the target. The Microprocessor situated in microcontroller replaced with in-circuit emulator which provides for processing, testing and debugging. The Emulator connected with all the rest of the components exactly in the same way connected to the Microprocessor. Debugging can be done using the emulator exactly the way debugging of the code done on the HOST, The functions required for debugging such as examine the memory and register contents, setting the breakpoints, executing the code situated within the breakpoints, writing the trace of execution, etc., can be carried by the in-circuit emulator. Emulators allow single step of execution, and one can see the contents of the memory even when the Microprocessor fails for any reason. When a program fails for any reason, emulator provides the dump of execution taken place toll the time the program halted. The testing and debugging software, the dump and the trace stored in a separate memory called overly memory, which contained with the in-circuit memory. The program situated in HOST communicates with the testing and debugging software stored in the overlay memory. The HOST as such is connected to the TARGET using either RS232C interface or any other technique for interfacing. Test commands are sent to the emulator using the program resident on the HOST.

One can see the content of the memory and registers when the processors or the built-in peripherals crashes. The trace of the program obtained when a program crashes using the emulator. The reasons for the failures investigated using the dump of the emulator tracing back to the source code.

The architecture of the Microprocessor and the locator output built into the in-circuit emulator. Every emulator is designed using a macro language used for submission of the test cases to the emulators. The emulator can carry testing to report response time in terms of instructions executed or the number of bus cycles used. Emulators are also designed to execute start-up code and interrupt service routines written in assembly language. The testing for portability of the code can only be tested using the in-circuit emulators. It is also possible to test the built-in peripherals such as DMA, UART, etc., using emulators as the emulator have been built having prior knowledge of the built-in peripherals. The process flow followed for testing using in-circuit emulator is shown in Figure 9.

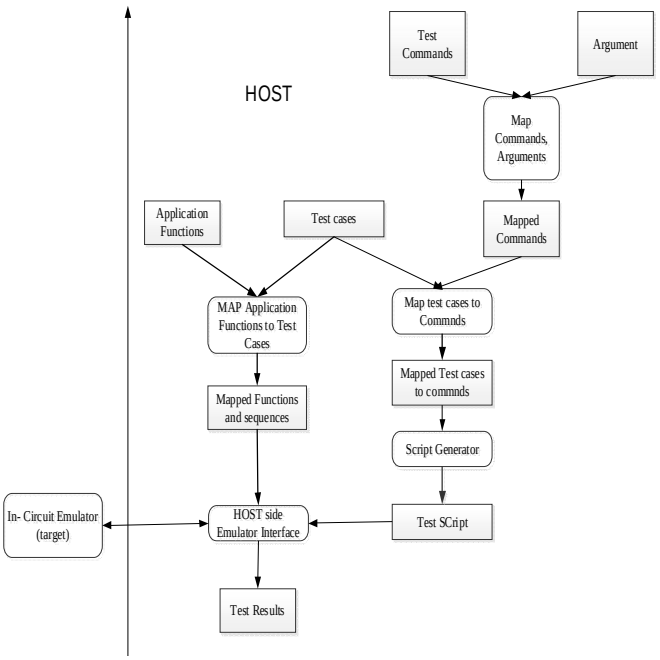

Figure 9: Process flow for undertaking to test at an Individual Location using the in-circuit emulator

Test cases that must be tested using the in-circuit emulator are first selected and classified location wise so that location-specific test cases can be selected. The test cases converted into test scripts that contain mapping test cases to commands and the related arguments, which represents the test cases used for undertaking testing. Test scripts stored in a file stored at HOST. The test scripts generated for the sample test cases shown in table 15.

Test Scripts are generated using the algorithm below:

Algorithm

Step-1

Reading test cases from test case repository that have been identified to be tested at a location using in-circuit Emulator

Step-2

Do the following process for each 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 to 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

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 executed

Step-5

Call the functions in the sequence required and at the end of completing execution of the entire functions store the test results

Step-6

Transmit the test results to the HOST

Step-7

Receive the test results (Table 16) and write the results to the Audit trail.

6.3.6 Testing through Logic Analyser

Different kinds of testing can be done using logic analyzers that include Testing proper working of hardware, validity,

and timing of the signals, the sequence of occurrence of the signals, timing of the signal received at a specific projection, the throughput of a signal, etc. Logic analyzers are used for testing, the proper working of the hardware. Testing connecting several logic analyzers at different locations is rather difficult. The hardware at different locations connected with a logic analyzer must be working for undertaking testing of the harder distributed to different locations. A test case may require proper working of the hardware at different locations.

Testing must be done to check other the individual embedded systems are communication properly as designed, especially when connected on networks using one serial, bus-based communication protocols. It is necessary to test whether the interfaces through which interconnection with the network achieved. The hardware interface provided at the individual embedded systems must be working properly for effecting the communication required. It is necessary to test whether the networking interfaces are in proper working condition while a communication request initiated. Undertaking such testing using logic analyzers is complicated.

Logic analyzers generally used for testing the hardware. The test case fed to Logic analyzers through commands and the test data fed as arguments to the commands both initiated from a PC. The logic analyzers after receiving the commands and the test data executes the test and send back the results to the PC where the results are stored, and audit trails are connected.

Hardware testing is done using test cases drawn from the database. The test cases converted into commands and command-line arguments which fed to the Logic Analyzers. The commands and command-line arguments transmitted to Logic Analyzer from a PC interfaced through, one of the existing communication interfaces. Logic Analyzer, transmit back the test results to PC after executing the commands using the command-line data. Figure 10 shows the way testing is done using the Logic Analyzers.

The test cases used for testing using logic Analyzer represented as test scripts. The process on the HOST side reads the test scripts one by one, and issues command along with the arguments to a logic analyzer. The commands executed by the logic analyzer, and the test results are netback to the Host. The test scripts generated shown in Table 17. Testing is carried as per the process flow using test scripts and the test results captured are shown in Table 18

6.3.7 Integrating the Test results

Testing hardware and software undertaken at each of the locations using the test cases designated to a particular location. Testing at each location is done using different kinds of methods such as scaffolding, assert macros, instruction set simulators, and instruction set emulators, and logic analyzers. The test results obtained at each of the locations are combined and Grouped such that test results related to a specific master test cases collated and merged to form overall test result that one will get if tested considering the entire distributed embedded system as a whole. Table 19 shows the integrated test results

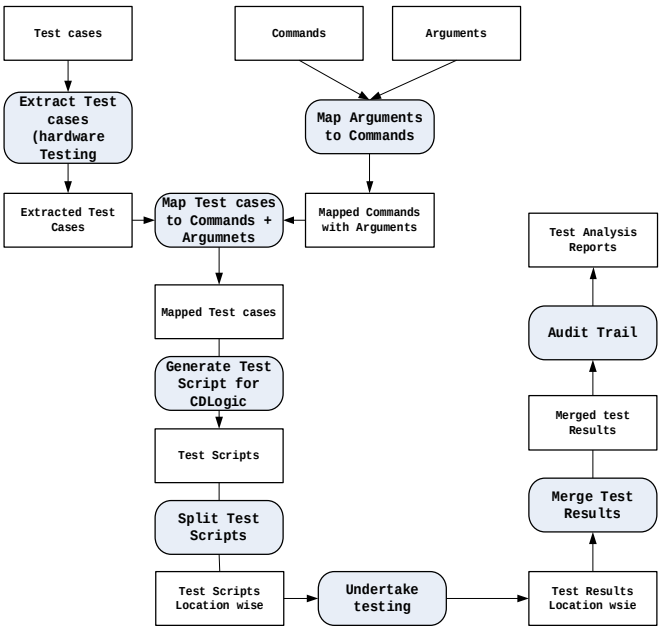

Figure 10: Testing through Logic Analyzer

Merging test results

The test results produced due to testing a group of test cases that relate to same master test case are merged to find the overall test result containing a master case that should have been tested considering the entire distributed embedded system as a whole.

The Merged test results will reveal buggy areas either in hardware or software are both — the test results used for assessing the overall reliability of the entire distributed embedded system. Table 20 shows the overall test results obtained for the pilot distributed embedded system. The overall reliability of the system with the test cases used appears to be nearing 100%.

7. CONCLUSION

Many application that ranges from health to space management implemented using distributed embedded systems. Some distributed embedded system might have sub-nets within a distributed embedded network. A distributed embedded system can be achieved either through a wired or wireless network. Several embedded systems which are heterogenous are connected using a communication system to achieve a distributed embedded system considering the requirements of distributing hardware and software. The way a distributed system functions is dependent on the kind of communication system used, which include I^2C, CAN, USB, RS485, etc.

Testing of a distributed embedded system is complicated as the environment required for undertaking testing varies largely. The testing environment is very much dependent on the method used for undertaking testing at each of the distributed locations. The testing of a distributed embedded system as such is complicated as it requires proper functioning of the communication system, and there is a need to set up the proper environment at each of the locations. Testing distributed embedded system considering the entire system is quite complicated.

No specific method has been in vogue for testing the distributed embedded system making it necessary to investigate the processes, tools, techniques that cater for testing a distributed embedded system achieved through wired communication of a specific type.

A prototype project is developed, which is meant for monitoring and controlling temperatures within a nuclear reactor system. The prototype model used as an experimental setup for proving the investigations and findings. A stand-alone embedded system converted into a distributed system through the distribution of hardware and firmware. The distributed embedded system has been designed using the communication systems that include I^2C, CAN, USB, and RS485. Testing of a distributed embedded system involves testing at the HOST, on Target and considering both HOST and the TARGET. Several methods that include scaffolding assert macros, instruction set simulators, in-circuit emulators, and Logic analyzers used for the undertaking of testing of distributed embedded systems. The hardware-independent code can be tested on the HOST using the methods scaffolding, assert macros and instruction set simulators. The testing of the hardware can be conducted using Logic Analysers. The Hardware dependent code can be tested using both HOST and the TARGET and In-circuit emulation method.

These days, most of the Microcontrollers provided with the RS232C communication interface. Sometimes, the controller boards provided with CAN, I2C, RS485 communication interface also, programmed for effecting communication. If the native communication interface does not exist, then the converters are used for converting RS232C to the expected communication interface — many methods used for allocating addresses to the computing nodes connected into a network. Communication is done using the addresses.

The standard methods such as scaffolding, Assert Macros, etc. that are in existence as on today using which the testing of standalone embedded systems carried needs to be modified, and new methods invented for undertaking the testing of distributed embedded systems. A single test case tested across the entire distributed embedded system divided into many test cases such that each of the test cases generated tested at a specific individual embedded system Location. The test results obtained at each location can be integrated through mapping and merging to get the overall status of testing the entire distributed embedded system. The testing of a distributed embedded system through Scaffolding method and networking of embedded system through the RS485 interface have has been presented. 80% of the testing can be carried using the scaffolding method as 80% of firmware is hardware-independent and therefore can be tested on a HOST.

The USB protocol system provides for an enumeration stage at which time the information related to the devices sent to a master for its complete understanding of the device. A new descriptor has been added using which the priority of messy messages fed to the slave and to make the slave store the same within it and use the same for checking the correctness of the message flow so that slave can inform the master if wrong message flow for any reason initiated**.**

Proper Hardware environment must be in existence carrying any function by the embedded system. The existence of an environment includes the existence of proper signals, the occurrence of the signals in a proper sequence, valid data, and data to be in a specific range, etc. In the absence of proper hardware environment, the embedded system shall certainly produce wrong results. The standard methods such as scaffolding, Assert Macros, etc., that are in existence as on today using which the testing of standalone embedded systems carried must be modified and new methods invented for undertaking the testing of distributed embedded systems. A single test case that tested across the entire distributed embedded system broken into several test cases that each of the test cases generated tested at a specific individual

embedded system. The test results obtained at each location integrated through mapping and merging get the overall status of testing the entire distributed embedded system.

Assert macros effectively employed for testing the existence of the environment required for a specific function carried further. Assert Macros help to test 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 the handling of an exceptional condition.

Every application requires, the flow of messages in a specific sequence so that coordinated processing takes place. For effecting communication as per a protocol, a certain type of packets must flow in sequence. The communication system must adhere to both flows of messages as per the application requirement and the movement of data packets as per the protocol requirements.

The standard methods such as instruction set simulators, scaffolding, Assert Macros, etc. that 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 distributed embedded systems. The test cases used for testing entire embedded systems broken into test cases can be tested at different locations and merge the results obtained out of testing at individual locations and merge the test results to get the overall picture of the testing entire distributed embedded system.

Some kinds of testing can be undertaken on the HOST simulating the way the code executed on the TARGET. Test cases related to testing throughput, response time, code and data conversion considering coding standards, parity, endian, world length, addressing modes, etc. can be undertaken through instruction set simulator. Instruction set simulator can be used for testing through simulation of the functioning of the hardware while at the same considering the heterogeneous issues. The testing through instruction set simulators is achieved through breaking and merging process, which is used as the basis for the testing an entire distributed embedded system.

The I2C communication system is the most frequently used system for networking heterogeneous embedded systems — reasonable speeds of communication achieved through I2C protocol. The communication system must be designed specifically suiting I2C protocol taking into considering addressing, arbitration, synchronizing, timing, etc.

I2C system does not prescribe any standard flow of the messages. However, the messages must flow as per the application requirements. The addresses assigned to the devices does not dictate the message flow. The data structures used for effecting communication are also different. A standard system of messaging designed and developed; the system used every time communication is to be done using I2C.

For achieving testing of entire testing, Hardware tested in addition to testing firmware. Test gadgets such as Logic Analyzers required for undertaking testing hardware. Many types of testing carried when it comes to hardware. The test cases include testing for device status, validity, sequencing, and timing of the signals. Testing is done to measure whether the desired response time and throughput achieved. It is necessary to mix the results at different locations and merge the results such way that the results so obtained reflect overall testing results considering the entire as a whole.

Considering the entire system and carrying the testing as a whole is complex and complicated due to several reasons, especially the complication involved in setting the environment required for undertaking testing. It is not possible to guarantee the proper working of the entire distributed system as a whole. Testing becomes complicated if hardware or communication interfaces fails.

Decomposing test cases into elementary test cases such that the test cases can be executed locally and then combining the test results will provide a basis and framework for undertaking testing of an embedded system as a whole. The Test results obtained at a different location can be ordered based on master test case wise, and in the order of subtest, case tested at one location. An algorithm developed for merging the test results which are obtained through testing at different locations and produce a unique representation of the same, reflecting the overall effect of undertaking the testing of the entire distributed embedded system.

REFERENCES

1. Chen-Huan Chiang, Paul J. Wheatley, Kenneth Y. Ho, Ken L. Cheung, Testing and Remote Field Update of Distributed Base Stations in a Wireless Network, IEEE Conference Publications, 2004, page no.711-718
2. Dae-Hyun Kum, Joonwoo Son, Seon-bong Lee and Ivan Wilson, Automated Testing for Automotive Embedded Systems. IEEE Conference Publications, 2006, page no. 4414-4418
3. Eric Armengaud, Andreas Steininger, Efficient Stimulus Generation for Testing Embedded Distributed Systems -The Flex Ray Example, IEEE, 2005, page no.763-770

4. G. Walters, E. King, R. Kessinger, R. Fryer. Processor design and implementation for Real-Time Testing of embedded systems. IEEE Conference Publications, vol.1, 1998
5. H. Thane, Real-Time Res. Center, Malardalen Univ., Vasteras, Sweden, H. Hansson, Towards systematic testing of distributed real-time systems, Real-Time Systems Symposium. Proceedings. The 20th IEEE, 1999, page no.360-369
6. J. Russell Noseworthy. The Test and Training Enabling Architecture (TENA) —Supporting the Decentralized Development of Distributed Applications and LVC Simulation. IEEE Conference Publications, 2008, page no. 259-268 https://doi.org/10.1109/DS-RT.2008.35
7. Pei Tian, JianchengWang, HuaijingLeng, Kai Qiang. Construction of Distributed Embedded Software Testing Environment. IEEE Conference Publications, vol.1,2009, page no. 470-473 https://doi.org/10.1109/IHMSC.2009.125
8. Peter H. Deussen, George Din, Ina Schieferdecker, An online Test platform for component-based systems. IEEE Conference Publications, 2002, page no.96-103
9. Sara Blanc, Pedro. J. Gil. Improving the multiple errors detection coverages in distributed embedded systems. IEEE Conference Publications, 2003, page no. 303-312
10. Silvie Jovalekic, Bernd Rist, Test Automation of Distributed Embedded Systems based on Test Object Structure Information, IEEE Conference Publications, 2008, page no. 343-347 https://doi.org/10.1109/EEEI.2008.4736543
11. Steven A. Walters, Practical Techniques for Distributed Real-time Simulation. IEEE Conference Publications, vol.2,1994, page no. 890-896
12. Tsai W. T., R Mojdehbakhsh and F. Zhu, Ensuring Systems and Software Reliability in the Safety-Critical Systems, IEEE ASET 98, Dallas, Texas, March 1998, page no.48-53
13. W. T. Tsai, L. Yu, A. Saimi. Scenario-Based Object-Oriented Test Frameworks for Testing Distributed Systems. IEEE Conference Publications,2003, page no.288-294
14. Yanfang Wang, Wandui Mao, Jinying Li, Peng Zhang, Xiaoping Wang, A Distributed Rectifier Testing System Based on RS-485. IEEE Conference Publications, 2010, page no. 779-781 https://doi.org/10.1109/ICIEA.2010.5515241
15. David E. Simon, An Embedded Software Premier, Pearson Publications, 1999, page no.313-319
16. The XYZs of Logic Analyzers Primer, Tektronix, A Logic Analyzer Tutorial part1, http://nutsvolts.texterity.com/, nutsvolts/200709/?folio=71&pg=71#pg71
17. Kyeongjoo Kim, Jihyun Song, Minsoo Lee, Real-time Streaming Data Analysis using Spark, International Journal of Emerging Trends in Engineering Research, Volume 6, No.1, 2018, pp. 1-5 https://doi.org/10.30534/ijeter/2018/01612018
18. Dr. J. Sasi Bhanu, Dr. JKR Sastry, B. Sunitha Devi, and Dr. V Chandra Prakash, Career Guidance through TIC-TAC-TOE Game, International Journal of Emerging Trends in Engineering Research, Volume 7, No.6, 2019, pp. 25-31 https://doi.org/10.30534/ijeter/2019/01762019
19. J. K. R. Sastry, A. Suresh, and Smt J. Sasi Bhanu, Building Heterogeneous Distributed Embedded Systems through RS485 Communication Protocol, ARPN Journal of Engineering and Applied Sciences, issue. 16, vol.10, 2015
20. Sastry JKR, J Viswanath Ganesh, Sasi Bhanu J, "I2C based Networking for Implementing Heterogeneous Microcontroller, based Distributed Embedded Systems", *Indian Journal of Science and Technology*, Vol. 8, Vol. 15, pp. 1-10, 2015-1 https://doi.org/10.17485/ijst/2015/v8i15/68739
21. Sastry JKR, Sai Kumar Reddy, Sasi Bhanu J, "Networking Heterogeneous Microcontroller based Systems through Universal serial bus," *International Journal of Electrical and Computer Engineering*, Vol 5, Iss. 5, 2015-2
22. Sastry JKR, Vijaya Lakshmi Machineni, Sasi Bhanu J, "Optimizing Communication between heterogeneous distributed Embedded Systems using CAN protocol," *ARPN Journal of engineering and applied sciences*, Vol. 10, Iss. 18, Pg. 7900-7911, 2015-
23. JKR Sastry, T. Naga Sai Tejasvi and J. Aparna, Dynamic scheduling of message flow within a distributed embedded system connected through RS485 network, ARPN Journal of Engineering and Applied Sciences, VOL. 12, NO. 9, MAY 2017
24. K. Chaitanya, Dr. K. Raja Sekhara Rao, Testing Distributed Embedded Systems Built over CAN using Scaffolding Method, International Journal of Emerging Technology and Advanced Engineering, issue.12, vol. 8 December 2018, page no. 28-42.
25. J.K.R. Sastry, K. Chaitanya, K. Rajasekhara Rao, D.B.K. Kamesh, An Effective Model for Testing Distributed Embedded Systems using Scaffolding Method, PONTE International Journal of Sciences and Research, issue.8, vol.73, 2017 https://doi.org/10.21506/j.ponte.2017.8.1

26. K. Chaitanya, Sastry JKR, K. N. Sravani, D. Pavani Ramya and K. Rajasekhara Rao, Testing Distributed Embedded Systems Using Assert Macros, ARPN Journal of Engineering and Applied Sciences, 2017, page no.3011-3021
27. Sastry JKR, K. Chaitanya, K. Rajasekhara Rao, DBK Kamesh, Testing Distributed Embedded Systems Through Instruction Set Simulators, PONTE, International Journal of Sciences and Research, issue.7, vol.73, July 2017, page no.353-382
28. JKR Sastry, K. Chaitanya, K. Rajasekhara Rao, DBK Kamesh, An Efficient Method for Testing Distributed Embedded Systems using In-circuit Emulators, PONTE, International Journal of Sciences and Research, issue.7, vol.73, 2017, page no.390-422
29. K. Chaitanya, JKR Sastry, K. Rajasekhara Rao, Testing Distributed Embedded Systems Using Logic Analyzer, International Journal of Engineering and Technology, March 2018, page no. 297-302.
30. Chaitanya Kilaru, K. Rajasekhara Rao, Comprehending Testing of distributed embedded systems, International Journal of Engineering and Technology, issue. 2.7, vol. 7 March 2018, page no. 303-307
31. K. Chaitanya, K. Rajasekhara Rao, Complication of Embedded Systems in Agriculture Technology using Customized Software, International Journal of Emerging Technology and Advanced Engineering, issue. 7, vol. 3, July 2013, page no. 368-373

Table 1: Comparison of existing methods

s.no	Author	Year published	Use of middleware	Existence of message flow system	Whether testing is automated /manual	Is testing done considering the whole system	Is communication system tested	What element is tested	Type of testing
1	Yanfang Wang, Wandui Mao, Jinying Li, Peng Zhang, Xiaoping Wang	2010	√	×	Manual	Whole	No	Communication devices	Device Status
2	Pei Tian, Jiancheng Wang, Huaijing Leng, Kai Qiang	2009	√	×	Manual	Whole	No	Environment	Environmental
3	Silvije Jovalekic, Bernd Rist	2008	√	×	Manual	Whole	No	Functional Interface	Integration testing
4	J. Russell Noseworthy	2008	√	×	Manual	Whole	No	Environment	Simulation
5	Dae-Hyun Kum, Joonwoo Son, Seon-bong Lee, and Ivan Wilson	2006	√	×	Manual	Whole	No	Functional	Manual
6	Chen-Huan Chiang, Paul J. Wheatley, Kenneth Y. Ho, Ken L. Cheung	2004	√	×	Manual	Whole	No	Functions	System
7	Sara Blanc, Pedro L.Gil	2003	×	×	Manual	Whole	No	Functions	Functional
8	W. T. Tsai, L. Yu, A. Saimi	2003	√	×	Manual	Whole	No	System	Regression
9	Peter H. Deussen, George Din, Ina Schieferdecker	2003	√	×	Manual	Whole	No	System	Conformance
10	G. Walters, E. King, R. Kessinger, R. Fryer	1998	√	×	Manual	Whole	No	Functions	Behavior
11	Steven A. Walters	1994	√	×	Manual	Whole	No	Functions	Functional
12	K. Chaitanya	2017	√	√	Automatic	Individual	Yes	Hardware Independent Functional testing, Environment	Functional

Table 3: Repository for the message flow system

Serial Number of device	Type of device	Device model number	Device Address	Port Number	Arbitration number	Message	Message Priority
1.	Master	LPC2148	50	1	11010101000	Master has the priority over the salves	1
2.	Slave-1	89C51	60	2	11010101001	Temp-1 flow before other messages	2
3.	Slave-2	AT89S52	70	3	11010101011	Temp-2 must follow temp-1 in a fraction of 10µsec	3
4.	Slave-3	PIC18F4550	40	4	11010100001	Message to pump-1 must follow temp-2 within 20µsec	4
5.	Slave-4	ATmega328	30	5	11010100011	Message to pump-2 must follow the message to pump-1 within 10µsec	5

Table 4 :Framework for the generation of test cases – Testing distributed embedded systems

Functional requirement Number	Functional Requirement	Test Requirements	Test Case serial	Sub-Test case serial	Split test cases	Testing Method	Testing Location
1	Read Temp-1 and write to LCD	Test Temp-1 read is written to LCD	1			Scaffolding	ES-1
		Test for proper sensing of Temp-1 signal at the output of the temperature sensor	2			Assert Macros	ES-1
2	Test the Communication based communication between the 89C51 (ES-1) and the central ES-5)	Test for the equivalence of output data sent (output Register) through ES-1 resident communication interface and received at the communication port of ES-5 (Comm. Input register)	3	3A	Test for proper outputting data on the output communication port of ES-1	Scaffolding	ES-1
				3B	Test for proper data read at the input communication on output communication port of ES-5	Scaffolding	ES-5
3	Read-Temp-1 and send to Central Micro Controller	Test for the equivalence of output data sent through the ES-1 resident communication interface and received at the communication port of ES-5	4	4A	Test for Reading of a particular Temperature-1 @ES-1	Scaffolding	ES-1
				4B	Test for Reading of a particular Temperature-1 @ ES-5	Scaffolding	ES-5
4	Read Temp-1 and measure throughput	Test for several times the temperature read in one minute	5		Test for throughput	Instruction set Simulator	ES-1
			6		Test for throughput	Scaffolding	ES-1

Table 5: Environment setting for undertaking Testing of a distributed Embedded System through Scaffolding

Command Serial	Command	Arg1	Arg2	Arg3	Arg4
1.	COMM-1				
2.	COMM-2				
3.	COMM-3				
4.	COMM-4				
5.	THRU-1				
6.	THRU-2				
7.	RESP-TEMP-1				
8.	RESP-TEMP-2				
9.	RESP-PUMP-1-ON				
10.	RESP-PUMP-1-OFF				
11.	RESP-PUMP-2-ON				
12.	RESP-PUMP-2-OFF				
13.	BZR-ON	TEMP-1	35	TEMP-2	30
14.	BZR-OFF	TEMP-1	35	TEMP-2	34
15.	PUMP-1-ON	TEMP-1	30	REFTEMP-1	25
16.	PUMP-1-OFF	TEMP-1	30	REFTEMP-1	32
17.	PUMP-2-ON	TEMP-2	30	REFTEMP-2	25
18.	PUMP-2-OFF	TEMP-2	30	REFTEMP-2	32
19.	WLCD-1	TEMP-1	30		
20.	WLCD-2	TEMP-2	35		
21.	RANGE-1				
22.	RANGE-2				
23.	RESP-BUZZER-ON				
24.	RESP-BUZZER-OFF				
25.	PUMP1-ON-RESPONSE				
26.	PUMP1-OFF-RESPONSE				
27.	PUMP2-ON-RESPONSE				
28.	PUMP2-OFF-RESPONSE				

Table 6 :A sample Test environment

Functional requirement Number	Test Requirements	Test Case serial	Sub-Test case serial	Command to be used	Testing Method	Testing Location
1	Test Temp-1 read is written to LCD	1	1	WLCD-1	Scaffolding	ES-1
2	Test for the equivalence of output data sent (output Register) through ES-1 resident communication interface and received at the communication port of ES-5 (Comm. Input register)	3	3A	COMM-1	Scaffolding	ES-1
3	Test for the equivalence of output data sent through the ES-1 resident communication interface and received at the communication port of ES-5	4	4A	COMM-1	Scaffolding	ES-1
4	Test number of times temperature read in one minute	6	6	THRU-1	Scaffolding	ES-1
6	Test whether PUMP-1 is ON when Temp-1 > Reference Temperature-1	8	8A	RANGE-1	Scaffolding	ES-1
7	Test whether PUMP-1 is off when Temp-1 <= reference Temperature-1	9	9A	RANGE-1	Scaffolding	ES-1
8	Test whether BUZZER is ON when ABS(Temp-1 -Temp-2)>2	10	10A	RANGE-1	Scaffolding	ES-1
9	Test whether BUZZER is OFF when ABS(Temp-1 -Temp-2)<=2	11	11A	RANGE-1	Scaffolding	ES-1
10	Test response time considering reading of Temp-1 and starting the pump-1 when Temp-1 > Reference Temp-1	12	12A	RESP-TEMP-1	Scaffolding	ES-1
11	Test response time considering reading of Temp-1 and stopping the pump-1 when Temp-1 <= Reference Temp-1	16	16A	RESP-TEMP-1	Scaffolding	ES-1
12	Test response time considering the reading of Temp-1 and Temp-2 and starting the buzzer when the difference between Temp-1 and Temp-2 >2	20	20A	RESP-TEMP-1	Scaffolding	ES-1
13	Test response time considering the reading of Temp-1 and Temp-2 and stopping the buzzer when the difference between Temp-1 and Temp-2 <=2	24	24A	RESP-TEMP-1	Scaffolding	ES-1
21	Test whether BUZZER is ON when Temp-2 > Temp-1	37	37B	RANGE-1	Scaffolding	ES-1
22	Test whether BUZZER is OFF when Temp-2 <= Temp-1	38	38B	RANGE-1	Scaffolding	ES-1
8	Test whether BUZZER is ON when ABS(Temp-1 -Temp-2)>2	10	10B	RANGE-2	Scaffolding	ES-2
9	Test whether BUZZER is OFF when ABS(Temp-1 -Temp-2)<=2	11	11B	RANGE-2	Scaffolding	ES-2

Table 7: List of functions implemented on all Distributed Embedded Systems

Functions for the Application at Location 1		
Function Code	**Name of the Function**	**Function Description**
F01	init-LCD ( )	Initialization of LCD
F02	cmdtoLCD ( )	To send command to LCD
F03	dispdataLCD ( )	To display data on to LCD
F04	delay ( )	To make a task to go to wait for the state for some defined time
F05	dispstrLCD ( )	Display a string on LCD
F06	read1ADC ( )	Read data from sensor through ADC
F07	RS232C_read ( )	Read data from PC to ES1
F08	RS232C_write ( )	Write data from ES1 to PC
F09	timer0 ( )	Timer function
Functions for the Application at Location 2		
F10	init-LCD ( )	Initialization of LCD
F11	cmdtoLCD ( )	To send command to LCD
F12	dispdataLCD ( )	To display data on to LCD
F13	delay ( )	To make a task to go to wait for the state for some defined time
F14	dispstrLCD ( )	Display a string on LCD
F15	read2ADC ( )	Read data form temperature sensor from ADC
F16	RS232C_read ( )	Read data from PC to ES2
F17	RS232C_write ( )	Write data from ES2 to PC
F18	timer0 ( )	Timer function

Table 8: Function calling sequence

Test Case serial	Sub-Test case serial	Split Test Requirements	Script	Function calls
1	1	Test Temp-1 read is written to LCD	1,1,WLCD-1, TEMP-1 , 30, ,	read1ADC()
7	7A	Test for proper outputting data on output communication port of ES-3	5,7A,COMM-3, , , ,	txstrES5ES3 ( ) txstrES3ES5 ( )
8	8C	Test whether the PUMP-1 is ON	6,8C,PUMP-1-ON, TEMP-1, 30, REFTEMP-1,25	setpump1on ( )
16	16C	Test for Response time of Pump-1 when off	11,16C,RESP-PUMP-1-OFF, TEMP-1=30	txstrES5ES3 ( ) setpump1off ( )
34	34A	Test for proper outputting data on output communication port of ES-4	18,34A,COMM-4, , , ,	TxchrES4ES5() RcvchrES5ES4()
10	10C	Test whether the BUZZER status is on when ABS(Temp-1 -Temp-2)>2	8,10C,BZR-1-ON, TEMP-1, 30, TEMP-2,33	recvDataFromES1 ( ) recvDataFromES2( ) compareTemp1WithTemp2 ( )
11	11C	Test whether the BUZZER status is off when ABS(Temp-1 -Temp-2)>2	9,11C,BZR-2-OFF, TEMP-1, 33, TEMP-2,30	recvDataFromES1 ( ) recvDataFromES2( ) compareTemp1WithTemp2 ( )
12	12B	Test if temp-1 > reference temp-1	10,12B,PUMP-1-ON, TEMP-1, 30, ,	rcvchrPCES1 read1ADC() setpump1on ( )
16	16B	Test if temp-1 <= reference temp-1	11,16B,PUMP-1-OFF, TEMP-1,30, ,	rcvchrPCES1 read1ADC ( ) setpump1off ( )
20	20C	Test if absolute of Temp1 - Temp-2 > 2	12,20C,BZR-1-ON, TEMP-1, 33, TEMP-2,28	recvDataFromES1 ( ) recvDataFromES2( ) compareTemp1WithTemp2 ( )
20	20D	Test for Response time of BUZZER when ON	12,20D,BUZZERONRESPONSE , , , ,	recvDataFromES2 ( ) compareTemp1WithTemp2 ( )
24	24C	Test if absolute of Temp1 - Temp-2 > 2	13,24C,BZR-1-OFF, TEMP-1,30,TEMP-2,28	recvDataFromES1 ( ) recvDataFromES2( ) compareTemp1WithTemp2 ( )
24	24D	Test for Response time of BUZZER when OFF	13,24D,BUZZEROFFRESPONSE, , , ,	recvDataFromES2 ( ) compareTemp1WithTemp2 ( )
36	36B	Test whether the PUMP-2 status is off when TEMP-2 <= Reference Temperature-2	20,36B,PUMP2OFFRESPONSE, , , ,	recvDataFromES2 ( ) compareRef2WithTemp2( )
37	37C	Test whether the BUZZER status is on when ABS(TEMP-1-TEMP-2) > 2	21,37C,BZR-1-ON, TEMP-1,28, TEMP-2,30	recvDataFromES1 ( ) recvDataFromES2( ) compareTemp1WithTemp2 ( )
38	38C	Test whether the BUZZER status is off when ABS (TEMP-1 - TEMP-2) <= 2	22,38C,BZR-2-OFF, TEMP-1,33, TEMP-2,28	recvDataFromES1 ( ) recvDataFromES2( ) compareTemp1WithTemp2 ( )

Table 9: Generated test Scripts

Functional requirement Number	Test Case serial	Sub-Test case serial	Split Test Requirements	Script
1	1	1	Test Temp-1 read is written to LCD	1,1,WLCD-1, TEMP-1 , 30, ,
2	3	3A	Test for proper outputting data on output communication port of ES-1	2,3A,COMM-1, , , ,
4	6	6	Test for through put	4,6,THRU-1,TEMP-1, , ,
10	12	12A	Test for Response time of temp-1	10,12A,RESP-TEMP-1, , , ,
11	16	16A	Test for Response time of temp-1	11,16A,RESP-TEMP-1, , , ,
14	29	29	Test for through put	14,29,THRU-2, , , ,
6	8	8C	Test whether the PUMP-1 is ON	6,8C,PUMP-1-ON, TEMP-1, 30, REFTEMP-1,25
7	9	9C	Test whether the PUMP-1 is OFF	7,9C,PUMP-1-OFF,TEMP-1, 30, REFTEMP-1,32
19	35	35C	Test whether the PUMP-2 is ON	19,35C,PUMP-2-ON,TEMP-2,30, REFTEMP-2,25
20	36	36C	Test whether the PUMP-2 is OFF	20,36C,PUMP-2-OFF, TEMP-2, 30, REFTEMP-2, 32
6	8	8B	Test whether the PUMP-1 status is on when TEMP-1 > Reference Temperature-1	6,8B,PUMP-1-ON, TEMP-1, 30, REFTEMP-1,25
7	9	9B	Test whether the PUMP-1 status is off when TEMP-1 <= Reference Temperature-1	7,9B,PUMP-1-OFF, TEMP-1,30,REFTEMP-1,32
8	10	10C	Test whether the BUZZER status is on when ABS(Temp-1 -Temp-2)>2	8,10C,BZR-1-ON, TEMP-1,30, TEMP-2,33
9	11	11C	Test whether the BUZZER status is off when ABS(Temp-1 -Temp-2)>2	9,11C,BZR-2-OFF, TEMP-1,33, TEMP-2,30
10	12	12B	Test if temp-1 > reference temp-1	10,2B,PUMP-1-ON, TEMP-1,30, PUMP-1-ON,1
11	16	16B	Test if temp-1 <= reference temp-1	11,16B,PUMP-1-OFF,TEMP-1,30, PUMP-1-OFF,0
12	20	20C	Test if absolute of Temp1 - Temp-2 > 2	12,20C,BZR-1-ON, TEMP-1,33, TEMP-2,28
12	20	20D	Test for Response time of BUZZER when ON	12,20D,BUZZERONRESPONSE, , , ,
13	24	24C	Test if absolute of Temp1 - Temp-2 > 2	13,24C,BZR-1-OFF, TEMP-1,30,TEMP-2,28
13	24	24D	Test for Response time of BUZZER when OFF	13,24D,BUZZEROFFRESPONSE, , , ,
19	35	35B	Test whether the PUMP-2 status is on when TEMP-2 > Reference Temperature-2	19,35B,PUMP2ONRESPONSE, , , ,
20	36	36B	Test whether the PUMP-2 status is off when TEMP-2 <= Reference Temperature-2	20,36B,PUMP2OFFRESPONSE, , , ,
21	37	37C	Test whether the BUZZER status is on when ABS(TEMP-1-TEMP-2) > 2	21,37C,BZR-1-ON, TEMP-1,28, TEMP-2,30
22	38	38C	Test whether the BUZZER status is off when ABS (TEMP-1 - TEMP-2) <= 2	22,38C,BZR-2-OFF, TEMP-1,33, TEMP-2,28

Table 10 : Sample Test results - Testing through Scaffolding

Functional requirement Number	Split Test Requirements	Test Case serial	Sub-Test case serial	Command	Input Variable-1	Input Variable Value 1	Input Variable-2	Input Variable Value 2	Test Output Variable	Test Output Values	Expected Output Values	Test Pass/fail
1	Test Temp-1 read is written to LCD	1	1	WLCD-1	TEMP-1	30			TEMP1-LCD-DATA	30	30	P
2	Test for proper outputting of data on communication port of ES-1	3	3A	COMM1	TEMP-1	30			temp1ToES5	30	30	P
3	Test for Reading of a particular Temperature-1 @ES-1	4	4A	COMM1	TEMP-1	30			temp1ToES5	30	30	P
4	Test for throughput	6	6	THRU-1					TEMP1 Thruput	12	12	P
6	Test whether the temp-1 read is within the Range	8	8A	RANGE-1					TEMP1 Range	1	1	P
7	Test whether the temp-1 read is within the Range	9	9A	RANGE-1					TEMP1 Range	1	1	P
8	Test whether the temp-1 read is within the Range	10	10A	RANGE-1					TEMP1 Range	1	1	P
9	Test whether the temp-1 read is within the Range	11	11A	RANGE-1					TEMP1 Range	0	0	F
10	Test for Response time of temp-1	12	12A	RESP-TEMP-1					RESP-TEMP-1	10	10	P
11	Test for the Response time of temp-1	16	16A	RESP-TEMP-1					RESP-TEMP-1	12	10	F
12	Test for Response time of temp-1	20	20A	RESP-TEMP-1					RESP-TEMP-1	10	10	P

Table 11: Generating Assert Macros

Assert Macro	Test to be carried	Parameter List	Assert Macro
MACRO-1	Test for proper sensing of Temp-1 signal at the output of the Temp-1 sensor	TEMP-1	#define assert (TEMP1) if (TEMP1 == 0) bad_assertion ("Improper Temperatue-1 value", TEMP1);
MACRO-2	Test for proper sensing of temp-1 and Pump-1 to be on	PORT-1-ON	#define assert (PORT-1-ON) If (PORT-1-ON == 0) bad-assertion ("improper pump1 status");
MACRO-3	Test for proper sensing of Temp-1 and Pump-1 to be off	PORT-1-OFF	#define assert (PORT-1-OFF) If (PORT-1-OFF == 1) bad-assertion ("improper pump1 status");
MACRO-4	Test for sensing Temp-1, Temp-2 and Buzzer On condition	TEMP-1, TEMP-2, BUZZER-STATUS-ON	#define assert (TEMP1, TEMP2, BUZZER-STATUS-ON) if (ABS (TEMP1- TEMP2) > 2 AND BUZZER-STATUS-ON == "ON") bad_assertion ("Improper Buzzer Status," TEMP1, TEMP2, BUZZER-STATUS-ON);
MACRO-5	Test for sensing Temp-1, Temp-2, and Buzzer off condition	TEMP-1, TEMP-2, BUZZER-STATUS-OFF	#define assert (TEMP1, TEMP2, BUZZER-STATUS-OFF) if (ABS (TEMP1- TEMP2) <= 2 AND BUZZER-STATUS-OFF == "OFF") bad_assertion ("Improper Buzzer Status," TEMP1, TEMP2, BUZZER-STATUS-OFF);
MACRO-6	Test for proper sensing of Temp-2 signal at the output of the Temp-2 sensor	TEMP-2	#define assert (TEMP2) if (TEMP2 == 0) bad_assertion ("Improper Temperatue-2 value", TEMP2);
MACRO-7	Test for Temp-2 and proper asserting of Pump-2 signal to be on	PORT-2-ON	#define assert (PORT-2-ON) if (PORT-2-ON == 0) bad-assertion ("improper pump2 status");

Table 12: Sample Test results at different Locations by using Assert Macros

Functional requirement Number	Split Test Requirement	Test Case serial	Sub-Test case serial	Macro unused/ Command	InputVariable1	Input Variable Value1	InputVariable2	Input Variable Value2	Test Output Variable	Test Output Value	Expected Output Value	Test Pass/ Fail
1	Test for proper sensing of Temp-1 signal at the output of the temperature sensor	2	2	MACRO-1	TEMP-1	30			TEMP-1	TRUE	TRUE	P
6	Test whether proper Temperature-1 sensor signal is being active	8	8D	MACRO-1	TEMP-1	30			TEMP-1	TRUE	TRUE	P
7	Test whether proper Temperature-1 sensor signal is being active	9	9D	MACRO-1	TEMP-1	30			TEMP-1	TRUE	TRUE	P
8	Test whether proper Temperature-1 sensor signal is being active	10	10D	MACRO-1	TEMP-1	30			TEMP-1	TRUE	TRUE	P
9	Test whether proper Temperature-1 sensor signal is being active	11	11D	MACRO-1	TEMP-1	30			TEMP-1	TRUE	TRUE	P
21	Test whether proper Temperature-1sensor signal is being active	37	37E	MACRO-4	TEMP-1	30			TEMP-1	TRUE	TRUE	P
22	Test whether proper Temperature-1 sensor signal is being active	38	38E	MACRO-5	TEMP-1	30			TEMP-1	TRUE	TRUE	P

Table 13: Generated Test Scripts – Instruction set Simulator

Functional Requirement Number	Test Case serial	Sub-Test case serial	Test Requirements	Split test cases	Testing Location	Test Script
4	5	5	Test for number of times temperature-1 is read in one minute	Test for throughput of TEMP1	ES-1	4,5,THRUTEMP1, , , ,
10	15	15A	Test response time considering reading of Temp-1 and starting the pump-1 when Temp-1 > Reference Temp-1	Test for Response for Temp1	ES-1	10,15A,RESTEMP1, , , ,
11	19	19A	Test response time considering reading of Temp-1 and stopping the pump-1 when Temp-1 <= Reference Temp-1	Test for Response time for TEMP1	ES-1	11,19A,RESTEMP1, , , ,
12	23	23A	Test response time considering reading of Temp-1 and Temp-2 and starting the buzzer when difference between Temp-1 and Temp-2 >2	Test for Response for Temp1	ES-1	12,23A,RESTEMP1, , , ,
13	27	27A	Test response time considering reading of Temp-1 and Temp-2 and stopping the buzzer when difference between Temp-1 and Temp-2 <= 2	Test for Response for Temp1	ES-1	13,27A,RESTEMP1, , , ,
12	23	23B	Test response time considering reading of Temp-1 and Temp-2 and starting the buzzer when difference between Temp-1 and Temp-2 >2	Test for Response for Temp2	ES2	12,23B,RESTEMP2, , , ,
13	27	27B	Test response time considering reading of Temp-1 and Temp-2 and stopping the buzzer when difference between Temp-1 and Temp-2 <= 2	Test for Response for Temp2	ES2	13,27B,RESTEMP2, , , ,

Table 14: Sample Test, results at different Locations using Instruction set simulators with commands and arguments

Functional requirement Number	Split Test Case	Test Case serial	Sub-Test case serial	Command	Test Output Variable	Test Output Values	Expected output value	Test Pass/fail
4	Test for throughput	5	5	THRUTEMP1	THRUTEMP1	10	10	P
10	Test for Response time of temp-1	15	15A	RESTEMP1	RESTEMP1	10	10	P
11	Test for Response time of temp-1	19	19A	RESTEMP1	RESTEMP1	10	10	P
12	Test for response time of temp-1	23	23A	RESTEMP1	RESTEMP1	10	10	P
13	Test for response time of temp-1	27	27A	RESTEMP1	RESTEMP1	10	10	P

Table 15: Test Scripts at Location-1 using In-circuit Emulator

Functional requirement Number	Test Requirements	Test Case serial	Sub-Test case serial	Split test cases	Testing Location	Test Script
10	Test response time considering reading of Temp-1 and starting the pump-1 when Temp-1 > Reference Temp-1	14	14A	Test for Response time of temp-1	ES-1	10,14A,RESP-TEMP-1, , , , ,
11	Test response time considering reading of Temp-1 and stopping the pump-1 when Temp-1 <= Reference Temp-1	18	18A	Test for Response time of temp-1	ES-1	11,18A, RESP-TEMP-1, , , , ,
12	Test response time considering reading of Temp-1 and Temp-2 and starting the buzzer when difference between Temp-1 and Temp-2 >2	22	22A	Test for response time of temp-1	ES-1	12,22A, RESP-TEMP-1, , , , ,
13	Test response time considering reading of Temp-1 and Temp-2 and stopping the buzzer when difference between Temp-1 and Temp-2 <=2	26	26A	Test for Response time of temp-1	ES-1	13,26A, RESP-TEMP-1, , , , ,

Table 16 : Sample Test results at different Locations using In-Circuit Emulator

Functional Requirement Number	Test Case serial	Sub-Test case serial	Split Test Case	Command	Input Variable-1	Input Variable-1 Value	Input Variable-2	Input Variable-2 Value	Test output Variable	Test output	Expected output	Test Fail/Pass
10	14	14A	Test for a Response time of temp-1	RESP-TEMP-1					RESP-TEMP-1	10	10	P
11	18	18A	Test for a Response time of temp-1	RESP-TEMP-1					RESP-TEMP-1	10	10	P
12	22	22A	Test for a response time of temp-1	RESP-TEMP-1					RESP-TEMP-1	10	10	P
13	26	26A	Test for a Response time of temp-1	RESP-TEMP-1					RESP-TEMP-1	10	10	P

Table 17 : Generated Test Scripts

Functional requirement Number	Test Requirements	Test Case serial	Sub-Test case serial	Split test cases	Testing Location	Test Script
10	Test response time considering the reading of Temp-1 and starting the pump-1 when Temp-1 > Reference Temp-1	13	13A	Test for the Response time of temp-1	ES-1	10.13A,RESPONSE-P1, , , ,
11	Test response time considering the reading of Temp-1 and stopping the pump-1 when Temp-1 <= Reference Temp-1	17	17A	Test for the Response time of temp-1	ES-1	11,17A,RESPONSE-P1, , , ,
12	Test response time considering the reading of Temp-1 and Temp-2 and starting the buzzer when the difference between Temp-1 and Temp-2 >2	21	21A	Test for the response time of temp-1	ES-1	12,21A,RESPONSE-P1, , , ,
13	Test response time considering the reading of Temp-1 and Temp-2 and stopping the buzzer when the difference between Temp-1 and Temp-2 <= 2	25	25A	Test for the Response time of temp-1	ES-1	12,25A,RESPONSE-P1, , , ,
12	Test response time considering the reading of Temp-1 and Temp-2 and starting the buzzer when the difference between Temp-1 and Temp-2 >2	21	21B	Test for the response time of Temp-2	ES-2	12,21B,RESPONSE-P2, , , ,

Table 18 : Sample Test results at different Locations using Logic Analyzer

Functional Requirement Number	Test Case	Test Case serial	Sub-Test case serial	Split test cases	Command	Input Variable 1	Input Variable Value1	Input Variable 2	Input Variable Value 2	Test Output Variable	Test output	Expected output	Test Fail/Pass
10	Test response time considering reading of Temp-1 and starting the pump-1 when Temp-1 > Reference Temp-1	13	13A	Test for Response time of temp-1	RESPOSNE -T1						10	10	P
11	Test response time considering reading of Temp-1 and stopping the pump-1 when Temp-1 <= Reference Temp-1	17	17A	Test for Response time of temp-1	RESPOSNE -T1						10	10	P
12	Test response time considering the reading of Temp-1 and Temp-2 and starting the buzzer when the difference between Temp-1 and Temp-2 >2	21	21A	Test for the response time of temp-1	RESPONSE -T1						10	10	P
13	Test response time considering the reading of Temp-1 and Temp-2 and stopping the buzzer when the difference between Temp-1 and Temp-2 <= 2	25	25A	Test for the Response time of temp-1	RESPONSE -T1						10	10	P

Table 19 : Combining and grouping-sample test results

Functional requirement Number	Functional Requirement	Test Requirements	Test Case serial	Sub-Test case serial	Split test cases	Command / Macros used	Input Variable-1	Input Variable Value 1	Input Variable-2	Input Variable Value 2	Test Output Variable	Test Output Values	Expected output	Test Pass/fail	Location	Test Type
1	Read Temp-1 and write to LCD	Test Temp-1 read is written to LCD	1	1	Test Temp-1 read is written to LCD	WLCD -1	TEM P-1	30			TEMP1-LCD-DATA	30	30	P	ES-1	Scaffolding
		Test for proper sensing of Temp-1 signal at the output of the temperature sensor	2	2	Test for proper sensing of Temp-1 signal at the output of the temperature sensor	MACR O-1	TEM P-1	30			TEMP-1	TRUE	TRUE	P	ES-1	Assert Macro
2	Test the Communic ation based communica tion between the 89C51 (ES-1) and the central ES-5)	Test for the equivalence of output data sent (output Register) through ES-1 resident communication interface and received at the communication port of ES-5 (Comm. Input register)	3	3A	Test for proper outputting data on the output communication port of ES-1	COM M1	TEM P-1	30			temp1ToES5	30	30	P	ES-1	Scaffolding
				3B	Test for proper data read at the input communication on output communication port of ES-5	COM M1	TEM P-1	30			temp1ToES5	30	30	P	ES-5	Scaffolding
4	Read Temp-1 and measure throughput	Test for throughput temperature read in one minute	5	5	Test for throughput	THRU TEMP 1					THRUTEMP 1	10	10	P	ES-1	Instruction Set Simulator
			6	6	Test for through put	THRU-1					TEMP1 Thruput	12	12	P	ES-1	Scaffolding

Table 20: Sample Merged Test results

Functional requirement Number	Functional Requirement	Split test cases	Command	Input Variable-1	Input Variable 1 Value	Input Variable-2	Input Variable 2 Value	Test Output Variable	Test Output Values	Expected output	Test Pass/fail	Overall test status
1	Read Temp-1 and write to LCD	Test Temp-1 read is written to LCD	WLCD-1	TEMP-1	30			TEMP1-LCD-DATA	30	30	P	PASS
		Test for proper sensing of Temp-1 signal at the output of the temperature sensor	MACRO -1	TEMP-1	30			TEMP-1	TRUE	TRUE	P	
2	Test the Communication based communication between the 89C51 (ES-1) and the central ES-5)	Test for proper outputting data on the output communication port of ES-1	COMM1	TEMP-1	30			temp1ToES5	30	30	P	PASS
		Test for proper data read at the input communication on output communication port of ES-5	COMM1	TEMP-1	30			temp1ToES5	30	30	P	
4	Read Temp-1 and measure throughput	Test for throughput	THRUTEMP1					THRUTEMP1	10	10	P	PASS
		Test for throughput	THRU-1					TEMP1 Thruput	12	12	P	