CHAPTER-1

INTRODUCTION
1.1 Introduction to Embedded Systems

1.1.1 An overview on testing embedded systems

Embedded Systems belong to different class of systems that differ greatly from the Loaded Systems in the following ways:

- Embedded systems are event driven
- Has limited Hardware Resources
- Single application specific
- Has to meet real time Requirements
- Users are intolerable of buggy systems
- Provide Low cost solutions
- ES solutions are delivered in multiples
- Has no user interface
- Has no secondary storage
- Has no support for software development and testing
- Both battery driven and power driven

Testing and debugging embedded systems are difficult and time consuming for simple reason that the embedded systems have neither secondary storage nor user interface. The users are extremely intolerable of buggy embedded systems. As such there is no restart of embedded systems as in the case of loaded systems.

Embedded systems deal with external environment by way of sensing the physical parameters and must provide outputs that control the external environment in real time. In embedded systems, the issue of testing must consider both hardware and Software individually and also the integration of both software and hardware. The mall-functioning of hardware is detected through software failures.

The ES target system does not support the required hardware and software platform needed for development and testing the software. The software development
cannot be done on the target machine. The software is developed on host machine and then installed onto the target machine which is then executed.

Embedded software or firmware as used referred to embedded systems is the software that is permanently stored in Read Only Memory (ROM) or Programmable ROM (PROM). It is easier to change the firmware than hardware. Firmware is responsible for the behavior of the entire embedded application. The firmware interacts with the hardware that senses the physical environment and controls the devices through which external environment is controlled.

Systems that must be controlled in real-time failing which they may fail, and the applications stop functioning and may result into negative results. Generally, these types of systems fall into the category of hard real time systems. There is another class of systems that lead to the disasters if the sensing and controlling functions are not achieved in real times.

The test target system sits in the test environment. The test environment provides for the hardware and software and data that are necessary for testing the target system. The hardware may include hardware devices such as In-Circuit Emulators, Logic analyzers, Controllers, A/D Converter, Sensors, Actuators, and the software includes embedded application, agents, monitors, third party tools, debuggers, simulators, macros, test scripts, test cases and test results. Figure 1.1 shows the typical environment that can be used for testing an embedded system.

HOST is a personal computer at which the test software, test data, test results, test scripts, agents, compilers, cross compilers etc. are resident. The integrated development platforms are installed on the HOST. Hosts shall generally be interconnected with the target through a serial interface or a network interface to
achieve communication with the Target System. Figure 1.2 shows the interfacing of Target System to HOST System.

Figure 1.1 Test Environments

Figure 1.2 Interfacing HOST with Target

Embedded System must be connected to the Production System and then testing must be done without which any amount of testing done offline will not
guarantee the proper running of the embedded system along with the production system. Figure 1.3 shows the complete Interfacing of the Target system, Host system and the Production System.

![Diagram](image)

**Figure 1.3 Target interfacing with Production system and the HOST**

The testing environment is often artificially built to closely simulate the real environment in which the target is finally deployed. A simulated test environment is often reusable and costs less than the real environment. In certain circumstances, provision of real test environment is not practically feasible.

A real time system typically monitors and controls some aspects of its environment. The most important are the timing requirements associated with monitoring and controlling of each of the target system parameters. Similarly, there are computational requirements to monitor and control the target environment.

To test a safety critical system, we need to provide the stimuli through a hardware interface or software initiated. The Stimuli shall stimulate the system into certain state or process. Stimuli can be considered as an active input to the test target.
Stimulus happens at the boundary of the target system. The internal components of a target system are not considered stimulus.

Test cases specify the input conditions and the expected outputs. The outputs that get generated as a result of executing a test case are compared with the expected results to determine whether a test has gone through or otherwise. A test case consists of one or more test scenarios. At least one test process is associated with one test case.

The execution of a test case produces stimuli to the test target. Both the stimuli and the response from the test target are recorded as log files and used as the input to the test process. Test cases can be chained together to exercise certain features.

Testing of embedded system requires testing of hardware, software and both together. The testing process requires a set of testing methods and mechanisms. The testing is to be done on HOST computer, and the target board is to be tested with the testing process initiated from the HOST. Figure 1.4 shows the environment required for comprehensive testing of the embedded system.

Embedded systems are designed to implement a specific application and through several challenges in term of throughput, response, testing and debugging, reliability, program installations and resource management.

Testing an embedded system is complex, and the entire testing required as such cannot be undertaken on the target board. Again, testing the embedded system offline really does not guarantee the reliable operation of the system. Embedded systems must be tested along with the production system as some of the events occurring within the embedded systems can be simulated and therefore cannot be tested. [Sastry JKR, K. Rajasekhar 2005] have discussed in detail various challenges and the ramifications of embedded systems.
Figure 1.4 Comprehensive testing of the embedded system

Embedded systems are built with few resources as they are primarily low-cost solutions and the resources such as secondary storage devices, user interface in terms of monitor, key board, mouse etc. are not available. The support of HOST based system is taken to test the embedded system that too when comprehensive testing of the embedded system is undertaken.

Embedded System Software must be testable before testing can be carried; embedded systems software must meet the following criteria to consider the software as testable.

- The System is decomposed into simple testable components.
- Outputs produced out of the system must be observable for each of the set of inputs.
- Understanding and documenting of the interface and dependencies between internal, external and shared components.
• Well organized technical documentation.
• Well documented change management.

A set of testing strategies must be adapted for testing an embedded system comprehensively. The following are some of the testing strategies that can be used for testing an embedded system.

• Testing of each module component wise.

• Conduct smoke testing in which unit testing is carried as soon as a component is ready and integration testing is carried as soon as a set of components are ready.

• Conduct Functional testing by using one or more of the testing methods which includes White Box testing, Black Box testing, Environment testing, Comparison testing and Orthogonal testing.

• Conduct Validation testing which is carried to determine the following:
  o Performance testing which typically involves determining the response time achieved for each of the External events.
  o Deviations from the requirements.
  o Checking the adoption of Security measures such as overflow conditions or exception conditions etc.

• The validation testing can either be carried at developer site (Alpha Testing) or at user site (Beta Testing)

• Recovery testing to test recovery of the system when either hardware or software fails

• System testing using either Top-Down, Bottom-Up or by using Regression approaches.

1.1.2 ES Testing Phases

Testing of Embedded systems is generally carried in three phases.

• In Phase-1 testing is carried completely in the Host machine at the time of development of the software on the Host Machine itself.

• In Phase-2 testing of the hardware is carried which can be carried parallel to Phase-1.

• In Phase-3 testing of the ES system is carried along with the Host which may be connected at a remote location.

To carry testing in any of the testing phases, test cases are required which must be identified using a framework.
1.1.3 The ES Verification Testing Types

There are many verification techniques and most of them fall into the category of dynamic and static testing.

**Dynamic testing**

Dynamic testing is carried when the system is under execution. A number of test cases are chosen, test data for each of the test case is determined and fed as input to the system and the output results are recorded and verified whether they confirm to the input. Dynamic testing can be further classified into Functional testing, Structural testing and Random testing.

**Functional testing**

Functional testing involves identifying and testing all the functions of testing which are defined within the requirements. Functional Testing is carried through black box testing since it involves no knowledge of the implementation of the system.

**Black Box Testing**

Black Box testing deals with event-based testing. Black Box testing is carried by choosing appropriate set of input conditions and generates events that test the functional validity. Black box testing means testing the behavior and performance. In an embedded system, Black box testing is carried to uncover the following:

- Discovering whether the system response is sensitive to particular inputs and System malfunctioning at these inputs and events.
- How the system behaves when the values of the inputs are beyond the set boundaries?
- Identifying the combination of inputs or objects or events that deal with system malfunctioning.
- Identifying the data rates of the inputs that modify the system performance.
Black Box testing is primarily done to test the relationships between the objects. The testing is primarily done for carrying boundary value analysis and not for all ranges of inputs. Each set of nodes in the logical paths are examined to describe functions and relationships.

**Structural testing**

Structural testing is carried with full knowledge of the implementation of the system which is done by carrying the White Box testing. White Box testing is carried by using the information from the internal structure of a system to device tests to check the operation of individual components.

**White Box Testing**

Every embedded application comprises two different types of tasks. While one type of task deals with emergent processing requirements, the other type of tasks undertakes the processing of input/output and also various processing related to housekeeping. The tasks that deal with emergent requirements are initiated for execution on interrupt.

All the tasks can be tested individually by way of conducting unit testing. Unit testing is a close examination of tasks in terms of all the logical paths. Testing is done in such a way that all the nodes are traversed at least once and also all the loops are tested and also all logical paths are traversed. The input design is made in such a way that all the logical paths are tested thoroughly.

Unit testing is done early in the development phase. As the numbers of logical paths are not too many in the embedded systems, unit testing can be conveniently employed.
**Environment Testing**

The environment testing related to testing an embedded system relates to the following:

- User interface testing when an LCD Display or Matrix Keys are present as a part of the embedded system. The testing is done to verify the effect of each command from the key and for the proper display of the messages.
- Testing Client-Server architecture is achieved through testing the server behavior for various kinds of inputs initiated from the client. Testing is also done to verify whether the response from the server is as per the specification expected from the Client.
- Testing help utilities.
- Testing the documents to ascertain whether the functions are working as per the documentation provided for the same.

**Comparison Testing**

Several available versions of the software are tested to find whether the later versions are proving the improved responses over the previous versions.

**Orthogonal Testing**

Testing is conducted for a combination of occurrence of events.

**Random Testing**

Random testing is carried by choosing the test cases randomly among the set of all possible test cases. The use of randomly determined inputs can detect faults that go undetected by other systematic testing techniques. Exhaustive testing is a form of random testing where the input test cases consist of every possible set of input values. Exhaustive testing performed at every stage in the life cycle results in complete verification of the system, it is realistically impossible to accomplish.

**Static Testing**

Static Testing does not involve the operation of the system or component. Several techniques exist for carrying static testing. While some of the techniques are manual, some are automatic.
The static testing techniques test the consistency of the software and also help measuring the properties of the software which include error proneness, understand ability and well-structured [Andiriole 1986].

1.1.4 The ES Validation Testing Methods

Validation testing is usually undertaken at the end of development cycle and looks at complete system as opposed to verification which focuses on smaller subsystems. Several validation techniques exist which include Formal Methods, Fault Injection, and Dependability analysis, Hazard analysis, and Risk analysis.

**Formal Methods**

Formal methods use mathematical and logical techniques to express, investigate and analyze the specification, design, documentation and the behavior of both hardware and software.

**Fault Injection**

Fault injection is the intentional activation of faults by either hardware or software means to observe the system operation under fault condition. Fault can be injected either through hardware or software. Hardware faults can be injected by physical intervention and the software faults are injected by manipulating the memory. Software fault injection is basically a simulation of hardware fault injection.

**Dependability, Hazard and Risk Analysis**

Dependability analysis involves identifying hazards and then proposing methods that reduce the risk of occurrence of the hazards. Hazards analysis involves using guidelines to identify the hazards, their root causes and possible counter measures. Risk analysis takes the hazards analysis further by identifying the possible consequences of each hazard and their probability of occurrence [Kopetz 1997].
1.1.5 The need for Verification and validation of embedded systems

Malfunctions in mission critical and safety critical systems such as nuclear reactor systems can cause havoc and lead to costly refinement and the malfunctions must be avoided [Tsai1998]. Investigation and techniques such as inspections, walkthroughs, and design reviews can be used for recognizing the potential variation during the requirements and design segment of the product improvement [Elliot 1994, Onama 1998, and Fagan 1976]. However, software testing is one of the critical phases of the product development where the product is ensured for its fulfillment with its requirements through its execution [Zhu 2002, Bezier 1990].

Software plays very important role of controlling the system in many safety critical systems through actuators. In-depth testing of the software is necessary when higher levels of reliability and availability of the embedded system is critical.

Software design, implementation and testing are performed iteratively with errors identified during test case execution. The errors are addressed by making changes to the software design and implementation is retested. Validation of embedded system is equally important as it involves the hardware testing through fault injection techniques and testing of response time and throughput as a whole.

Embedded applications belong to a different class of applications which throw several challenges especially related to testing. The Software testing process to test the embedded applications involves testing individually hardware, software and both hardware & software together. The process of testing an embedded application is rather complex and needs a detailed study. The process of testing a loaded application is more streamlined and several of the tools such as Load Runner, WIN Runner, Team Test etc., are available using which different testing can be carried in a unified manner.
1.1.6 The need for comprehensive testing of the embedded systems

Enlargement and testing of embedded software is especially complex because it typically consists of large number of concurrently executing and interacting tasks. Each task in embedded software is executed at different intervals under different conditions and with different timing requirements. Furthermore time available to develop and test embedded software is usually quite limited due to relatively short lifetime of the products.

Cost-Effective testing of embedded software is critical in maintaining a competitive edge. Testing an embedded system manually is quite time taking and also a costly preposition. Tool based testing of an embedded system has to be considered and put into use to reduce the cost of testing and also complete the testing of the system rather quickly.

Tools such as Windriver, Tarnado, Tetware etc. are also available in the market for testing embedded application but they carry fragments of testing and even the fragments’ of the testing is not done in unified manner. The tools fail to address the integration testing of the components on the target machine and also in-between the components and the software that is resident on the Target system.

In embedded systems, the issue of testing must consider both hardware and software. The mall-functioning of hardware is detected through software failures. The target embedded system does not support the required hardware and software platform needed for development and testing the software, hardware and both. The software development cannot be done on the target machine. The Software is developed on host machine and then installed in the target machine which is then executed.
The Goals for testing the embedded systems differs from that of the loaded systems. Testing of embedded systems directly using target board will not realize all the testing goals of the embedded systems.

- Finding the bugs early in the development process is not possible as the target machine often is not available early in the development stage or the hardware being developed parallel to software development is either unstable or buggy.

- Exercising all the code including dealing with exceptional conditions in a target machine is difficult as most of the code in embedded systems is related to uncommon or unlikely situations or events occurring in timing relationship with other.

- Difficulty to develop reusable and repeatable tests as it is to create repeatable event occurrence sequence in the target machine.

- Maintenance of audit trail related to test results, event sequences, code traces, and core dumps etc. which are required for debugging cannot be maintained in the target machine due to primarily not having any disk storage in target machine.

To realize the testing goals, it is necessary that testing be carried in the host machine first and then be carried along with the Target [J.K.R. Sastry, K. Rajasekhar et al., 2007]. Embedded Software must be of highest quality and must adapt to excellent strategies for carrying testing. In order to decide on the testing strategy or the type of testing carried or the phases in which the testing is carried, it is necessary to carry on with the analysis of different types of test cases that must be used for carrying the testing.

Test cases must be identified sufficiently enough that all types of tasks that constitute the application must be tested. Several types of testing such as unit testing, integration testing, END-TO-END testing, Regression testing etc. are also to be conducted to test the embedded Systems as in the case of loaded systems.

Embedded systems are event driven initiated due to a change in the physical environment as a consequence of a stimuli. Events occur in some sequence and there
may be relationships existing between the occurrences of the events. Events also provide the end user perspective. Embedded system must be tested for proper processing of the events right from the stage of event occurrence due to a stimulus till the time the control action is taken. The testing of the event-based processing can be considered as integration testing.

Events are processed by using patterns. Patterns are related to the timing and occurrence of prior and post conditions. A Pattern of processing may require the fulfillment of some initial conditions and some output conditions after the event has taken place. The prior conditions and the post conditions must occur as per the stated timings. The processing of the events as per pre-identified patterns is equally important. The testing of the embedded systems must be undertaken from END-TO-END testing which is a kind of integration testing. END-TO-END testing must be undertaken from the user perspective. It is the minimal integration testing required to ensure that events are processed as per the patterns attached to them.

Changes also take place after the embedded system is implemented. Changes may take place in hardware or software or both. The Regression testing of the embedded systems must also be undertaken every time a change is undertaken.

Testing embedded systems must be undertaken either offline or online. Offline testing of embedded system is conducted by simulating the production system. Online testing is undertaken when the embedded system is connected to the production system. The testing should be undertaken either offline or online depending on the requirement.

The testing of the embedded system software can be conducted at the HOST using techniques such as Scaffolding, Assert Macros, or Instruction Set Simulators. The testing of the embedded system hardware can be carried using the test gadgets
such as logic analyzers connected to the hardware and probing for the occurrence of various kinds of faults. The logic analyzers can be driven by HOST based software to which the gadget is connected. The test gadget in this connects the TARGET and HOST.

The testing of the behavior of the production system when connected to the embedded system can be carried along with the HOST machine. The primary strategy of testing should be that the testing be carried as fast as possible and as reliable as possible as the testing can be carried at different locations and using different methods.

Testing embedded systems is complex. Testing embedded systems must be done quite early in the development of the system. Development of the embedded systems involves analysis, design and development of both hardware and software in individual paths by using V model. Testing hardware and software is done independently and then are integrated and tested together.

Testing software must be done considering the whole code instead of individual units to guarantee proper running of the software. The testing strategy must include various locations of testing, and also different methods used for the testing.

Comprehensive testing of the embedded systems involves conducting the Unit testing, Integration testing, END-TO-END testing, and Regression testing. The testing process should consider all aspects of testing hardware, software and both. The comprehensive testing of the embedded system involves the usage of several methods, tools, techniques and locations. [J.K.R. Sastry, Rajasekhar et al., 2007] have proposed a test process architecture model for undertaking the integrated testing that will help in undertaking the comprehensive testing of the embedded system.
1.2 Overview on testing embedded systems

The evolution of embedded systems is throwing many challenges from time to time especially when the safety and mission critical systems are to be built using embedded systems. The issue of fault handling has to be addressed to a greater length. Modern computer processor is often delivered with errors, while intelligent hardware subsystems may exhibit non-deterministic behavior. Operating systems and programming languages are becoming increasingly complicated and their implementations less trustworthy. In addition component based multi-tier software system architectures increase the number of failure modes, while internet connectivity exposes systems to malicious attacks. Blind reliance on standards can provide developers with false sense of security.

Planning in advance for the new challenges is as important as embracing the new technology. Despite the important advances made over the last decades in the area of software engineering and the successful realization of many safety-critical software systems, the evolution of computer technology is creating new challenges especially in dealing with different types of failure modes.

Several challenges are to be addressed while implementing different types of embedded systems. The challenges include throughput, response, testability, debug-ability, reliability, memory management, power management etc. One of the main challenges that one has to face is to test the embedded systems from different perspectives.

Software for embedded systems is written for meeting various functions that an embedded system must support. Most of the functions of an embedded application must be met without any human intervention. Embedded software has to deal with number of uncommon events, which can sometimes occur simultaneously. Testing of
embedded system is difficult as it is not easy to generate uncommon events that occur simultaneously.

As embedded systems are resource constraint, there is no way available for an embedded system to report the user when bug crops up. When a bug occurs, any embedded system simply stops working. Debugging software does not exist to find the reason for the bug. Special techniques are required for debugging the embedded software. No embedded system is allowed to crash. The embedded systems must be fault tolerant and the fault tolerance can be implemented by hardware redundancy and exception handling mechanisms within the software implementation.

An embedded system has to work without the need for rebooting or resetting, typical of many desktop systems. This calls for very reliable hardware and software. For example, if an embedded system comes to halt because of a hardware error, then system should reset itself without the need for human intervention. Certainly, reliability is critical in any embedded system.

Embedded systems have become a pervasive part of daily life used for tasks ranging from providing entertainment to assisting the functioning of key human organs. Embedded computer systems are used to automate critical real-time applications, including process control systems etc. While, these mission critical embedded applications raise obvious reliability concerns, unexpected or premature failures in these systems can erode the wide spread acceptability of new systems.

Generally, embedded systems are cost sensitive and often work with limited resources such as smaller memory size or diskless designs. These constraints can make it difficult to apply many traditional computer design methodologies for improving. Embedded systems reliability, providing reliable embedded systems operation while satisfying constraints such as power consumption, real-time power
consumption and real-time throughput. Embedded systems often reside in machines that are expected to run continuously for years without errors and in some cases recovery is to be affected if any error occurs. Therefore, the software is usually developed and tested more carefully than that for personal computers, and unreliable mechanical parts such as disk drives, switches or buttons are avoided.

Embedded systems cannot safely be shutdown for repair or it is too inaccessible to repair. Many subsystems with redundant spares that can be switched over to or software “limp modes” that provide partial function may have to be implemented. Examples include space systems, cables, navigational beacons, borehole systems and automobiles. The Embedded systems must be kept running for safety reasons. “Limp modes” are less tolerable; often backups are selected by an operator. Examples include aircraft navigation reactor control systems, safety-critical chemical factory controls, train signals engines on single-engine aircraft. The systems that use embedded systems will lose large amounts of money when shutdown, telephone switches, factory controls, bridge and elevator controls, funds transfer and market making, automated sales and service etc. Since embedded systems are to be fault tolerant and the failure of any part of embedded system shall lead to failure of the entire embedded system, they are tested thoroughly by conducting different types of hardware and software testing.

Several issues are to be addressed when it comes to testing the embedded systems which include location of testing, methods used for testing, gadgets used for testing etc.

Failures in mission critical and safety critical systems such as nuclear reactor systems can cause havoc and lead to costly rectification and the failures must be avoided [Tsai1998]. Analysis and design techniques such as inspections,
walkthroughs, and design reviews can be used for identifying the potential inconsistencies during the requirements and design phases of the product development [Elliot 1994, Onama 1998, and Fagan 1976]. However, software testing is one of the critical phases of the product development where the product is checked for its compliance with its requirements through its execution [Zhu 2002, Bezier 1990].

Embedded systems are low cost solutions and therefore the time and cost spent for testing the embedded systems must be as least as possible. One way to reduce the cost of testing is to conduct minimal testing. The life cycle model must be such that minimal testing will ensure high reliability of the developed embedded system.

Embedded Systems deals with external environment by way of sensing the physical parameters and also must provide outputs that control the external environment.

To realize the testing goals of the embedded systems, it is necessary that testing be carried in the host machine first and then be carried along with the Target [J.K.R. Sastry, K. Rajasekhar Rao et al., 2007]. Embedded Software must be of highest quality and must adapt to excellent strategies for carrying testing. In order to decide on the testing strategy or the type of testing carried or the phases in which the testing is carried, it is necessary to carry on with the analysis of different types of test cases that must be used for carrying the testing. Comprehensive testing of the embedded systems requires the usage of test cases and various architectural frameworks and models, tools and processes.

[K. Rajasekhra Rao, 2009] has recommended several models for undertaking the comprehensive testing of the embedded systems. He has recommended that comprehensive testing of the embedded system is a necessity when testing of
hardware, software and both using several distinct methods at different testing locations.

There are varieties of test case types that must be considered for undertaking the testing of the embedded systems. There is a need for classification of test case types as hardware test case types, software test case types and the test case types related to both hardware and software.

Various architectural frameworks and models for undertaking the testing of embedded systems comprehensively have been presented by K. Rajasekhra Rao. The architectural models proposed include the models related to configuring and designing the test environment, designing and architecting the semantics of the models, E2E process models, the testing process models, models for generating tests cases, models for undertaking the actual testing etc.

The architectural models for capturing the test requirements from the end user perspective have been presented. The building of scenarios provides the basic framework for identifying the testing requirement from the perspective of any given application. The process of constructing complex scenarios from simple scenarios has also been presented.

A framework for modeling the testing of an embedded system from the point of view of E2E has been modeled. Every path that exists in embedded system application that includes both hardware and software elements are modeled through E2E. The relationships of thin threads with tasks, functions, methods have been identified and built into the central repository.

The scenarios are linked to thin threads as the thin threads provides for integration testing of an embedded system. K. Rajasekhra Rao proposed maintenance of a repository that stores the scenarios, the relationships between the scenarios, and
the relationship of scenarios with thin threads. The repository helped in constructing the repeated audit trails.

The process models required for undertaking the generation of test cases and undertakings testing at each of the locations have been clearly identified. The sequence of processing undertaken right from compilation stage to the test case generation and testing of the embedded system has been presented. All integration aspects to integrate the hardware, software, test gadgets and production system have also been presented.

He has recommended that a repository be built that stores all the elements of the testing and repository forms the basis for automatic generation of the test cases and use the same for undertaking the testing through adaption of different methods of testing.

The process of automatic generation of test cases has been explained. All the test case types that help comprehensive testing of embedded system are mapped to the thin threads and the test cases required for undertaking testing through each of the test methods have been presented.

The repository is also populated with the analysis and design elements, such as statements, functions, tasks, data variable types, data variables, functions call sequencing etc. The relationships between all the design elements have also been built and populated into the database.

The events that happens and the sequence in which the events happen are modeled as patterns. The pre and post conditions that should be satisfied for a pattern to happen have been presented.

Thus, the central repository is built based on the architectural frameworks and test case generators are used to generate the test cases that should be used for testing
the embedded system at different locations and using different test methods. The process models using which the actual testing is carried are presented. The way the generated test cases are used for undertaking testing by using different methods at different locations have been presented.

Thus, the models presented by K. Rajasekha Rao have helped testing the embedded systems comprehensively and thus lead to development of high quality embedded systems. The models as such lacked the guarantee of undertaking testing from different domain perspectives.

1.3 Overview on Combinatorial Methods

Generating test cases for testing software comprehensively has become complex as the number of possible input combinations can exceed the number of atoms in the ocean. It is difficult to show that the program works correctly for all inputs. Combinatorial test case generation methods offer a solution. Combinatorial test case generation methods analyze interactions among variables and help to detect errors early in the life cycle of the development of the software.

Combinatorial testing aims at finding the minimal tests cases that are enough to test systems comprehensively. The test cases are generated based on input combinations or output combinations or the combination of both input and output variables. The combinatorial testing is employed to detect possible t-way combinatorial interactions for t=2, 3, 4, 5, 6 or more. Different combinatorial strategies have been presented in the literature during 1985 and 2017.

Verification and validation of highly-configurable software systems, such as those supporting many input parameters or customizable options, is a challenging activity. Combinatorial Interaction Testing (CIT) techniques can be effectively applied, in practice, to various systems including embedded systems. CIT consists of
employing combination strategies to select values for inputs and combine them to form test cases. These tests can then be used to check how the interaction among the inputs influences the behavior of the original system under test. The most commonly used combinatorial testing approach is to systematically sample the set of inputs in such a way that all t-way combinations of inputs are included. This approach exhaustively explores t-strength interaction between input parameters, generally in the smallest possible test executions. For instance, pair wise interaction testing aims at generating a reduced size test suite which covers all pairs of input values.

There are a number of ways a system can be tested and there are a number of automatic test case generation tools available, but these suffer from combinatorial explosions to test all possible test scenarios. Combinatorial strategies are useful for undertaking testing of many applications.

Many strategies exist to apply combinatorial methods for undertaking the testing of complex systems. It is necessary to study and understand the application of a strategy to a chosen application. A basis for comparisons among the combinatorial strategies has to be evolved and the comparisons are made so that the applicability of a strategy to an application can be decided. Combinatorial testing strategies are test case selection methods that identify test cases by combining values of different test object input parameters based on some combinatorial strategy.

The combinatorial testing has been addressed in several dimensions that include methodologies, designing methods, test constraints, modeling through UML diagrams, fault diagnosis, test case prioritization, metrics, tools, and case studies.

Combinatorial testing approaches can also be used for testing different applications which are developed using WEB, embedded systems, grid computing, cloud computing networking, event simulation, parallel processing etc.
1.4 Overview on use of Combinatorial Methods for testing Embedded Systems

Combinatorial methods mainly concentrate on reduced cost, complete testing, high quality, use of interaction rules, efficiency of time, controlling the risk. The main objective being generation of minimum or optimal test cases that when used for testing will comprehensively test a system under test.

Embedded systems must be tested rigorously or else a user throws away the buggy systems. Embedded systems must be high quality systems, the stimuli, the interaction between the inputs and outputs must be considered while deciding on what to test. Embedded system must be tested comprehensively and within minimal time meaning really quick.

A survey has been conducted [M. Lakshmi Prasad et al., – 2017] that gives comprehensively the status of use of combinatorial methods for testing the embedded systems and the combinatorial methods in general. The analysis revealed that combinatorial methods would help generating test cases using the external perspective considering the input, output, input-output and multi-output domain. However, the combinatorial methods do not consider any of the issues related to internal behavior. The combinatorial methods help in generating fewer test cases considering just the external behavior of the embedded systems. Thus it becomes necessary to expand the scope of combinatorial methods to include the internal behavior of the systems also so that comprehensive set of test cases can be generated using which systems like embedded systems can be tested thoroughly. Most of the methods presented in the literature have just shown the way test cases can be generated but not the way to be used for undertaking the actual testing of a system under test.
The scope and objectives of using combinatorial methods really matched the basic issues related to testing the embedded system making a point and reason to apply combinatorial testing methods for testing the embedded systems.

1.5 Problem definition

Testing embedded systems comprehensively is most crucial as the users reject buggy systems in no time. Testing embedded systems quite thoroughly has to be done especially when the embedded systems are used for monitoring and controlling safety critical and mission critical systems. Majority of the effort is spent on testing embedded systems nearing 50% of the total effort required for the development of the embedded systems. There is clear identification of both input and output domain and the relations that comprise both the domains giving a clear insight into the consideration of combinatorial methods for the generation of test cases. In addition the Input-output domain and Multi-output domain must also be considered to cover all the perspectives of the embedded systems. Embedded systems require a set of processes using which one can undertake actual testing of the embedded systems.

Combinatorial testing is a specification based sampling technique that provides a systematic way to select combinations of program inputs or features for testing. It is an effective testing technique to test hardware, software and both that reveal failures in a given system based on input, output, Input-Output, Multi-Output combinations. It has been applied over the years to test input data, configurations, protocols, graphical user interfaces, software product lines etc.

The combinatorial testing methods have been proved to be efficient compared to other methods as the methods lead to generation of fewer test cases using which testing is carried in minimal time. Considering the efficiency of these methods it
becomes necessary to investigate the use of combinatorial methods for undertaking testing of embedded systems.

This research aims at investigating and presenting innovative combinatorial methods that can be used for undertaking comprehensive testing of embedded systems and also provides a set of processes using which actual comprehensive testing of embedded systems can be undertaken.

1.6 Objectives

1. To invent effective methods in comparison with other existing methods that can be used for generating test cases considering the input domain and all variables relating to embedded systems (External + Internal, Continuous, Discrete and Binary).

2. To invent effective methods in comparison with other existing methods that can be used for generating test cases considering the output domain and all variables relating to embedded systems (External + Internal, Continuous, Discrete and Binary).

3. To invent effective methods that can be used for generating test cases considering the input-output domain and all variables relating to embedded systems (External + Internal, Continuous, Discrete and Binary).

4. To invent effective methods that can be used for generating test cases considering multi-output domain and all variables relating to embedded systems (External + Internal, Continuous, Discrete and Binary).

5. To invent effective methods that can be used for generating test cases considering multi-input domain and all variables relating to embedded systems (External + Internal, Continuous, Discrete and Binary).

6. To investigate, find and implement the methods using which actual comprehensive testing of embedded system can be undertaken.

1.7 Scope

1. Development of a proto-type embedded system that can be tested by using the newly invented methods considering all the four perspectives.

2. Conduct literature survey to find embedded system based existing test case generation methods.

3. Conduct literature survey to find actual methods used for undertaking testing of the embedded systems.

4. Conduct literature survey to find existing combinatorial methods that can be used for generating test cases considering input, output domain, and Input-Output and Multi-Output domain.
5. Carry gap analysis to find shortcomings of the existing methods in respect to generating and testing embedded systems.

6. Investigate and find methods that suits generating test cases considering each domain separately and keeping in view requirements of embedded systems.

7. Carry out the comparative analysis of the newly found methods with the existing methods and bring out various efficiencies and the reasons for using such methods.

8. To investigate and find the methods that can be used for undertaking actual testing of the embedded systems.

9. Test newly found methods using the prototype model and prove the sufficiency of the same.

1.8 Limitations

The scope of the research is limited to testing stand-alone embedded systems. The applicability of these algorithms is confined to testing embedded systems only. Testing distributed embedded systems is beyond the scope of this research work.

1.9 Chapterisation

The thesis has been organized into 8 Chapters. Chapter-1 Introduces embedded systems, issues related to testing embedded systems, combinatorial methods and the use of the same for generation of test cases that can be used for testing the embedded systems. Chapter-1 also includes the problem definition, scope, objectives and limitations of the research presented in this thesis.

Chapter-2 details an embedded system that implements an application that is related to monitoring and controlling the temperatures within nuclear reactor systems (TMCNRS). The application is tested by using the test cases that are generated using different methods that have been presented in the remaining chapters.

In Chapter-3, generation of test cases that can be used for testing the embedded systems using only the input domain has been explained. The method presented in the chapter has been compared with other existing methods in the literature. The combinatorial method presented in this chapter considered all specific
issues of the embedded systems unlike other methods existing in the literature. The method is developed using graph-based strategies. The method has been used for generating the test cases which can be used for testing the Prototype embedded system.

Generating test cases using output domain is yet another approach which has been used for generating test cases by many of the combinatorial methods which have not addressed specific issues of the embedded systems. Efficient test case generation method that generates the test cases using the output domain and specific features of the embedded system has been presented in this chapter-4 which is built using Particle Swarm Optimization (PSO) techniques.

The methods presented in chapter-3 and chapter-4 address only sufficiency of either input domain or exactness of the output domain. The adequacy of output generated when input domain is used for generating the test cases or coverage of all possible inputs when output domain has been used cannot be assured. Consideration of input-output domain and multi-output domains together supports the sufficiency and exactness. A Neural Network based method that takes into account the input-output domain has been presented in chapter-5. The method has been used for generating the test cases required for testing a prototype embedded system. Comparison of those methods with other methods has been made just considering input domain and the method is found to be effective and efficient.

Many of the embedded systems deal with array outputs which are either sequential or random and could be of different types. Dependency of one output over another output also exists. In chapter-6 generation of test cases considering multi-output domain has been presented. The grouping of the outputs into different vectors considering output characteristics of the embedded systems has been the basis used
for generating the test cases that can be used for testing the embedded systems. Comparison of this method with other methods has been made just considering input domain and the method is found to be effective and efficient.

Many of the embedded systems deal with array outputs which are either sequential or random and could be of different types. Dependency of one output over another output also exists. In chapter-7 generation of test cases considering multi-input domain has been presented. The grouping of the outputs into different vectors considering output characteristics of the embedded systems has been the basis used for generating the test cases that can be used for testing the embedded systems. Comparison of this method with other methods has been made just considering input domain and the method is found to be effective and efficient.

In chapter-8, the way the actual testing can be carried by using the test cases generated through any of the methods has been presented. It has been explained that every test case must be decomposed into elementary level such that a test cases can be tested using one of the relevant methods that includes methods such as Scaffolding, Assert Macros, Instruction Set Simulator, In-Circuit Emulator and Logic Analyzers.

Conclusions have been drawn in chapter-9 and directions to the future works have been detailed in this chapter.

1.10 Conclusions

In this chapter, the use of combinatorial methods and the importance of considering specific issues of embedded systems as a part of combinatorial methods for generation of test cases that can be used for testing an embedded system have been explained. The problem definition, scope, objectives and limitations presented in this chapter are all centered on generation of test cases for testing the embedded systems. The scope is limited to testing standalone embedded systems.