CHAPTER II

Comprehensive Requirements Specification for Testing of Embedded Systems
2.0 Comprehensive Requirement Specification for Testing an Embedded System

2.1 Introduction

Embedded Applications are 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 and Software together. The process of testing an embedded application is rather complex and needs a detailed study. The process of testing a loaded application however 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.

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

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

Many Tools are also available in the market for testing embedded applications to 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.

Testing and debugging embedded systems is difficult and time consuming for simple reason that the embedded systems have neither storage nor user interface. The users are extremely intolerable of buggy embedded systems. Embedded systems deal with external environment by way of sensing the physical parameters and also must provide outputs that control the external environment.

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

The Goals for testing the embedded Systems differs from that of the loaded systems. Testing of the embedded systems directly using target board is not sufficient to 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 of the code including dealing with exceptional conditions in a target machine is difficult as most of the code in an 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 difficult 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. Embedded Software must be of the 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.

Every embedded Application comprises of two different types of tasks. While one type of tasks deals with emergent processing requirements, the other type of tasks undertakes the processing of input/output and also various processing related to house keeping. The tasks that deal with emergent requirements are initiated for execution on interrupt. 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 stimulus. 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 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 form END-TO-END Testing which is a kind of integration testing. END-TO-END testing must be undertaken from the user perspective. The END-TO-END testing is a kind of Integration testing but from 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 under taken.

Testing of the 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 systems 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 Locations.

2.2 Types of Testing

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 is selected 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 on the host machine itself.

Block 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 range of inputs. Each set of nodes in the logical paths are examined to describe functions and relationships. Black Box testing is carried when the focus of testing is the functional requirements. Black Box testing is carried on the host along with the target.

The environment testing in 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 for proper display of signal occurrence on the PC monitor or logic analyzer display area
- Orthogonal testing is to be carried to verify the proper sequence of occurrence of events, parallel occurrence of the events and some times occurrence of uncommon events.

2.3 Testing Approaches

Several authors have proposed different approaches to conducting testing of embedded systems. [Jacobson 1999] and others have suggested testing of modules of embedded systems by isolating the modules at run time and improving the integration of testing. This method has however failed to support the regression of events.

[Nancy 2004] and others suggested an approach of carrying unit testing of the embedded systems using agile methods and using multiple strategies. Testing of embedded software is bound up with test of hardware, crossing professional and organizational boundaries. Even with evolving hardware in the picture, agile methods work well provided multiple test strategies are
used. This has powerful implications for improving the quality of high reliability systems, which commonly have embedded software at their heart.

[Tsai 2001] and others have suggested END-TO-END Integration testing of embedded system by specifying test scenarios as thin threads; each thread representing a single function. They have even developed a WEB based tool for carrying END-TO-END Integration Testing. [Lee NH 2003] suggested a different approach for conducting integration testing by considering interaction scenarios since the integration testing must consider sequence of external input events and internal interactions.

Regression testing [Tsai WT 2001] has been a popular quality testing technique. Most regression testing’s are based on code or software design, Tsai and others have suggested regression testing based on Test scenarios and the testing approach suggested is functional regression testing. Tsai and others have even suggested a WEB based tool to undertake the Regression testing.

[Jakobson et al 1999] and others have suggested testing of embedded systems by simulating the Hardware on the host and combining the software with the simulators. This approach however will not be able to deal with all kinds of test scenarios related to Hardware. The complete behavior of Hardware specially unforeseen behavior cannot be simulated on a host machine.

[Tsai 2003] and others have suggested a testing approach based on verification patterns, the key concept of this being recognizing the scenarios into patterns and applying the testing approach whenever similar patterns are recognized in any Embedded Application. But the key to this approach is the ability to identify all test scenarios that occur across all types of embedded applications.
Even though several authors have suggested several approaches to conducting the several types of testing, no attempt has been made to identify all possible test scenarios considering both Hardware and software.

2.4. Identification of Test Scenarios

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

2.4.1 Testing Hardware Independent Code

Embedded Systems Software development is done on a host as the Target machine will not have sufficient resources to undertake the software development. The following are the primary reasons:

- In adequate memory to store compilers, linkers, locators
- Lack of storage to store execution traces, event logs and test History
- Non availability of user Interface in terms of Key Board, Mouse and Display screen

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

If an analysis is carried on the entire code of an ES application it will reveal that 80% of code is hardware Independent code and the rest is Hardware dependent code. It makes it easy, faster, cost effective to complete testing of the Hardware Independent code on the Host Machine It self.

The testing of the Hardware Independent code can be carried at the HOST by carrying any of Techniques that includes Scaffolding, Assert Macros and Instruction set simulators.

2.4.1.1 Testing through Scaffolding:

Using this technique, Testing of Hardware Independent code can be carried on the Host Machine. The Scaffolding software is added to the code by replacing the Hardware dependent code and provides for the same entry points to the hardware independent code as provided in the he original code. Fig 2.1 shows the division of code.

Hardware Dependent Code and the code that runs within the Hardware must be replaced by another piece of Software called Scaffolding Software. The Scaffolding Software must provide the same entry points that a Hardware dependent code provides on the target Machine. The Scaffolding Software must call the same functions that relate to Hardware independent Code.
The Scaffolding software takes instructions from the keyboard or from a file existing on the disk and produces output onto the display or into a log file. The Scaffolding software must provide a clean interface with the hardware independent code.

When hardware independent code calls functions that communicate with Hardware then technically the code is not divided into hardware independent code and Hardware Dependent Code. If the functions that deal with the hardware are written in Scaffold software it will make the Scaffold software complicated. In the Scaffold Software all the Hardware Code is replaced by simulation software that simply simulates the functioning of the Hardware.
The Scaffolding software must replace all the input and output functions that deal directly with the hardware and must provide code that acts as if inputting and outputting is done through the functions. The Data Flow that must be undertaken to achieve the replacement of Input and output functions by Scaffolding software is shown in the Fig 2.2.

When an interrupt is initiated by the Hardware due to occurrence of an external event, the Hardware dependent code is processed in the interrupt service routine and a call to the function contained in the Hardware independent code is made to complete the entire processing. To isolate the hardware dependency, it is necessary that Scaffolding Software must execute the interrupt routine directly to simulate a situation that an interrupt has occurred. Scaffolding software must simulate the occurrence of various types of events.

Fig 2.2 Input/Output simulation of Hardware Device based Processing
The Process Flow that must be undertaken to call the interrupt service routines directly by the scaffolding software simulating a situation that an interrupt has occurred is shown in the Fig 2.3.

In the real time systems some of the processing to be done is based on passage of time. Generally the passage of time is implemented through a timer. The timing value is set by the program and the timer hardware device initiates an interrupt at which time its related Interrupt service routine is executed. To simulate this kind of processing the scaffolding software may use the clock based time by way of using the system calls. If passage of time is implemented in this way, the scaffolding software will loose track of managing time as many other processes may be executing concurrently when time delay is to be caused for a particular function. It is better that Scaffolding Software calls the Timer Interrupt routine to have a better control of the Timing of interrupts and also execute the tasks as per the timing requirements.

---

**Fig 2.3 Simulating the Interrupt service Routines by Scaffolding Software**
The Process Flow that must be undertaken to call the Timer related Interrupt routines directly by the scaffolding software simulating a situation that passage of time has occurred is shown at Fig 2.4

In a real time situation a task submits a request to be processed by a device by way of making an entry in its queue. The control tasks imitate a request to the hardware for processing the request. The hardware after processing the request initiates an interrupt and its related interrupt service routine executed. Interrupt service routines set the status of hardware and call the control task to process another request. This kind of a situation can be tested by scaffolding software capturing input released by control tasks and call the interrupt service routine directly.

The process Flow that must be undertaken to implement the interaction between the control task and the scaffolding software and the scaffolding software directly calling the interrupt service routine is shown in Fig 2.5
In a real time system, interaction between a set of devices is undertaken to accomplish processing of an external event. When the event is initiated by a device its related interrupt service routine is executed which in turn calls the device specific control task to process the event. The control task of the first device after processing the event communicates with the control task related to the second device by way of sending a message. The control task sends the message to the second device which will be processed by its driver. The second device after processing the message request initiates an Interrupt to indicate that processing of the message request is completed. To test this kind of situations the scaffolding software must call the interrupt service routine of the first device and also must capture data released by the second control task to the second device and then call the Interrupt routine of the second device to infer that processing of the message request to the second device is completed. The process Flow that must be undertaken to test the interaction between two devices by the Scaffolding software is shown in Fig 2.6

Fig 2.5 Scaffolding output to a device
Fig 2.6 Scaffolding the interaction between the devices

Some times some situations arise wherein several tasks initiate messages to a particular device for processing by way of making entries into the queue related to the devices. Control task related to the device process the requests for each of the task separately by way of outputting the messages to the device. The device after processing the request initiates an interrupt at which time its related interrupt service routine is executed. The control task will have the logic to time space the messages received from the same Task. To test this kind of scenario Scaffolding software must read the queue isolating the control task and call the interrupt service routine directly as if the processing of the output message is completed. The scaffolding software must implement time spacing of the messages from the same task on its own.
The Process Flow that must be undertaken to test the interaction between several tasks and a device and the Scaffolding software facilitating the testing in the absence of direct device interaction is depicted at Fig 2.7

A Situation arises when a request initiated by any of the devices through its Interrupt service routine is processed by any other device meant for controlling purposes which is set to communicate with the device. This kind of test scenarios can be tested by making all the Interrupt service routines call the scaffolding software and the scaffolding software calling the interrupt routine of a device whose characteristics are set matching the input device. The control tasks calls the scaffolding software to indicate the properties with which the output devices are set.

The Process Flow that must be undertaken to test the interaction between several devices which are set with the same characteristics using the scaffolding software is shown in Fig 2.8
The process of testing can be captured into a file and the scaffolding software reads the script file and parses each line in the script file and carries on with the testing by way of calling the tasks, control tasks and the interrupt service routines in the sequence required. The script file may contain the commands along with parameter value. Even scaffolding software can undertake the testing using a command line inputted from the Key Board.

Proper test cases should be generated and written to an output file based on the testing requirement specification. The testing requirement specification that should be considered for testing the Hardware Independent code could be stated as a set of scenarios such as given below.

- Testing various tasks bypassing the hardware dependent code and simulating the occurrence of an event.
- Testing the various tasks by way of simulation of Hardware dependent code
- Testing interaction with the peripheral devices bypassing the interrupt routines
- Testing the Time based functions by calling the Timer Interrupt service routines directly
- Testing tasks by way of simulating the occurrence of a particular event
- Testing the chain of tasks by way of simulating the occurrence of a sequence of events
- Testing the chain of tasks by simulating the simultaneous occurrence of a set of events
- Testing the tasks by simulating the occurrence of uncommon events
- Testing the Tasks for the failures of communication interfaces
- Testing the interaction of tasks with control task that deal with the devices by way of bypassing the interrupts initiated by the Hardware
Fig 2.8 Scaffolding Interaction of Several Devices to control a Hardware device such as an actuator

- Testing the process that several tasks initiate requests to be processed by a device and the device initiates an interrupt that processing is complete
- Testing repeated requests initiated by external interrupts for processing by the same device specially controlling the time delay between different types of processing
- Testing the communication between the multiple instances of hardware independent code
- Testing for the proper operation of resources such as Queues, Mail Boxes and Pipes
• Testing for the Overflow condition i.e. trying to write to the queue or mailbox or pipe when it is already fully written
• Testing for underflow condition i.e. trying to read a queue, or Mail Box or Pipe when that is empty

• Testing the startup code

• To test whether the requisite tasks are identified with the Real time operating system
• To Test whether memory pools are created properly
• To test whether the handles required related to queues, Mail Boxes, Pipes, events and Timing have been created within the startup code

2.4.1.2 Testing through Assert Macros

Testing of an embedded System at the HOST can also be carried by way of writing inline code using macros. Macros are supported by the cross compilers. A macro can test whether prerequisites are satisfied before proceeding with any of the execution. ASSERT is the most frequently used Macro which takes only one parameter as its argument.

Assert Macro evaluates the parameters for its TRUE or FALSE value. The macro does nothing if the parameter is evaluated to TRUE. If the parameter is evaluated to FLASE the Macro reports the error and causes the program to crash.

The assert macros can evaluate an expression submitted as a parameter in terms of the following:

• Null Pointer evaluation
• Validation for range of values
• To verify whether a function is called by ISR or a Task
To check the setting of the status bits

Assert Macros helps in conducting the Environment Testing by way of checking the availability hardware and software environment before proceeding with the regular code. Test case can be generated to test the environment testing. The scaffolding software should have a routine to read the Assert Macros position the same at appropriate locations and the compiled executable module can then be made to run. Any Environment problem could then be reported to an output file. The output generated can then be integrated with the assert Macros and then can be included into a log file using which the Audit Trail can be conducted. Fig 2.9 shows the integration of Scaffolding Software with Assert Macros.

Assert macros help bringing the bugs to light sooner and gives various clues about the problems existing in the code. Assert Macros cannot call functions such as exit () or abort () which terminates the program and transfer control to RTOS. If assert macros are to be functional on the target machine no outputting is done to the peripheral devices to report the error; instead may call the Bad Assertion function which may do the following.

![Diagram](Fig 2.9 Integrating Assert Macros with Scaffolding Software)
• Disable interrupts and spin into an infinite loop to stop the system.
• Turn on some unexpected LED or blink the LED to a characteristic rhythm.
• Write values to specified memory Locations.
• Write Location of the instruction being executed by way of reading the stack through an assembly code. The actual instruction that is resident at the address location can be known from the locator map or the Logic Analyzer can report the contents of the memory Location.
• Call a function or execute an illegal instruction that causes an emulator or target debugger to stop.

Using the assert Macros the following can be tested.

• Usage of proper Devices addresses specially MAC addresses.
• Checking for proper data bits related to the devices.
• Checking for proper setting of the value.
• Checking for Range of values to be contained in a Variable.
• Checking for proper Function Calling Sequences.

2.4.1.3 Testing through Instruction set Simulators:

The scaffolding software or Assert micros can really test the portability issues related to the code before the code is migrated to the Target. The Scaffolding and Assert Macros cannot test the following.

• Cannot test the direct interaction with the real Hardware
• Cannot test the response time and throughput as real interaction with the hardware is bypassed
• Cannot test the portability issues as the entire development and testing is carried in host and actually the code must run on a different processor after completing the testing and then the code is migrated to target machine.
The host machine differs from the Target processor in many ways which include the following.

- Significance of the Bytes (Endean)
- Processing 16 Bit and 32 Bit variables in an expression
- Type Casting
- Addressing
- Parity
- Floating point processing

- Cannot test the shared Data Problem as the real event based processing is not undertaken.

Instruction set simulators will be handy to test the response time, throughput and portability issues. Simulator is software that runs on 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.

The simulators can run a scaffold software and also can execute the assert Micros in addition to testing the image for various portability, throughput and response achievement etc. Fig 2.10 shows the integration of Instruction simulators with scaffolding Software.

Any of the readily available instruction set simulators can be used and interfaced with scaffolding software. The instruction set simulator can be configured to be interfaced with the Scaffolding Software.

The process of development of the image which will be subjected to an instruction set simulator is shown in the Figure 2.11
Simulators do the simulation by way of using the memory for registers, program counters, and address registers and data buffers. The instructions are read from the memory and converted to instructions equivalent to the target machine. Simulators provide user interface to provide the following debugging facilities.

- Running a program
- Setting break points
- Examine and change data in the memory and the registers
- Single step through the program

Simulators should support a Macro Language using which testing scenarios are submitted as input to the simulators. Simulators work in conjunction with Scaffolding Software and facilitate communication between them through memory.
Simulator can report response time in terms of the number of Target Machine instructions executed. The count of instructions executed or number of bus cycles used. The average response time can be computed by multiplying with average instruction execution time.
Simulators can also execute the startup code and interrupt service routines written in assembly language. Most of the testing related to Portability issues can be tested using the Simulators as the simulation of instructions of the target machine is done by the simulators. Simulators also help in testing the Built in peripherals such as timer, DMA, UART etc. as the simulation of such built in simulators is quite possible. Simulators have the prior knowledge of the target Processors and related Built-ins.

Simulators cannot however test the custom built Hardware.

The typical test case scenarios that can be tested using the Simulators are given below.

- Response time for each of the event based processing
- Response time for processing the events that occur simultaneously.
- Testing for Portability for each of the cases below.
  - Significance of the Bytes (Endean)
  - Processing 16 Bit and 32 Bit variables in an expression
  - Type Casting
  - Addressing
  - Parity
  - Floating point processing
- Testing the Built-in peripheral devices
  - Reading from ROM
  - Reading from RAM and writing into RAM
  - Peripheral Interface Testing through DMA
- Throughput based testing
  - Amount of time it takes for processing a set of events simultaneously.
When testing of the above scenarios is required it is necessary that test cases are generated and written to output file which can be read by the instruction set simulators and the test results are written to output files.

2.4.1.4 Testing through Third Party Tools:

At the host several types of testing can be conducted considering the quality of the Embedded Applications especially when bugs are to be investigated when ever the errors are traced while running the application on the HOST. The test cases are submitted to a third party tool and the third party tool conducts the testing using the image of Embedded Application and produces the test results back to test process which maintains the test results in the secondary stage using which the Audit Trail can be conducted.

Fig 2.12 shows HOST based Testing using third Party Tools. A Number of third party tools are available in the market using which several types of testing using the ES application can be made. The third party tools will be handy specifically for debugging the application when bugs are identified and the same are to be fixed.

Memory based testing

- Memory Leaks due to the call to OS
- Memory Leaks due to the calls to third party tools
- Testing for writing to the end of an Array
- Testing for usage of freed pointers
- Testing for proper Memory allocation without any overlapping as the system runs
- Writing to a Queue when it is full
- Reading from a queue when it is empty
- Record of memory allocation
- Graphical display of memory allocation and usage
Profile based Testing

- Testing functional interaction on a complete tree of functions
- Testing for Task Interactions
- Testing for the High Use routines
- Testing for stack allocation for every call
- Record Usage Histories
- Testing for race condition
- Testing for priority Inversion
- Testing for deadly embrace
- Testing for passing of parameters due to function calls
- Testing for multithreaded processing of functions
Variable based Testing

- Test for change of value of a Variable or set of variables and also recording of peak values of the variables
- Test for change of values of the variables by triggering certain events
- Test for variables by name when in scope, the stack, and registers of the target processor

Testing for coverage of code

- Testing for coverage of a functions
- Testing for coverage of a block of code
- Testing for coverage of Control Structures
- Testing for converge of Loop Structures

Testing for JTAG based debugging

- Testing for scanning of Multiple JTAG Device connections
- Testing for multiple debugging connections to JTAG Devices

Testing the functioning of the devices

- Testing for Hardware System bring up
- Testing for Faulty devices early in the development life cycle
- Testing of processor activity of bus devices

2.4.2 Testing the Hardware, Hardware Dependent Code Along with HOST

A section of the code is completely dependent on Hardware and cannot be tested without the Target Machine. Again Hardware dependent code cannot
be tested by way of only using the Target due to lack of resources required to carry testing. Therefore Testing of Hardware dependent code has to be carried along with the Host Machine.

Testing the hardware itself requires the integration of hardware testing gadget with the Target and the HOST machine as the gadget should be told what to test and how to test. The Hardware testing gadget connects to the Target through probe and with HOST using RS232C or RJ45 physical interface. Test cases are issued to Logic Analyzer through HOST and the Logic analyzer after conducting the Test shall make available the Test results to the HOST.

While 80% of code testing can be done on the host, the following types of testing cannot be however carried on the Host alone and therefore needs a testing process that uses both host machine and Target machine. The following are the different types of testing that need the usage of both Host and the Target machines.

- Signals based testing in terms of timing, sequence, validity and patterns
- Instruction fetching
- Memory reads and writes and the validity of values written and the appropriateness of the addresses used to fetch data and instructions
- The occurrence of events and the way the events are processed
- Response time of event based processing
- Recognizing the occurrence of uncommon events
- Timing of execution of the first instruction of an interrupt routine after an interrupt signal is initiated.

2.4.2.1 Testing the Hardware Along with the HOST using Logic Analyzers

Some of the testing required by embedded system cannot be tested by software alone. Hardware equipment like Logic Analyzers is required to carry
testing of complex mechanisms that need specific timing of the signals. The timing of various signals must be measured and presented in a graphical form. Some times the time duration during which the signals are valid must also be measured. For proper processing the timing of the signals and the sequence in which signals are processed must be tested. The Triggering of the signals whether edge triggered or level triggered and the voltage levels of the signals must be measured and stored in order to prepare and display timing of the signals. The occurrence of the events can be tested by way of sensing the change in the level or edge of a signal.

Logical analysis also help in computing the response time by way of measuring the time of two signals and finding the time difference. The pattern in which the signals are triggered is also important that proper computing is done. Using Logical analysis the patterns of signal processing can also be undertaken.

Logical analysis can be carried either in timing mode or state mode. In timing mode one of the available signals should be used as the reference signals for computing the timing of other signals where as in state mode a separate clock signal must be used as reference signals. Conducting Logical analysis in state mode one can test the way the instructions are fetched and the data read from memory or written into memory.

In timing Mode the following testing can be carried
  • Timing of the signals
  • Sequence of occurrence of signals
  • Pattern of occurrence of signals
  • Validity of the Signals

In state mode the following testing can be carried

  • To test whether proper addresses are fetched for instructions
- To check the kind of values being written to the memory addresses
- The state of signal when a clock input is either edge triggered or level triggered in terms of address in the address lines, data in the data lines, Read/Write on the control lines and Read enable/Write Enable Signals
- Storing the trace of instructions executed
- To check the time at which the first instruction of an interrupt routine executed after asserting a Interrupt signal by peripheral devices.
- To trace the uncommon events occurring through tracing and storing the signals
- To test the time durations during which a particular device is accessed by way of defining filters.
- To test the sequence of occurrence of a signals along with mapping of execution of the Embedded Systems Code to trace the areas of problem occurrence.

![Diagram](image)

**Fig 2.13 Integration of Logic Analyzers with HOST and the TARGET**

The signal based testing is normally carried using separate hardware and software with built-in hardware logic to sense the signals, store the signals
and process the signals. The special setup will have the facilities to store the instruction trace and also store the changed made to memory Location.

Logic analyzers are to be integrated with the HOST and the target machine to achieve the integrated testing requirements. Fig 2.13 shows the environment required for undertaking the Hardware Testing using Logic Analyzers.

The following types of testing cannot be carried with Logic Analyzers.

- Signals cannot be tested through executing the code by enforcing the break points
- Cannot be used to simulate the target machine in terms of registers and memory locations
- Testing for the contents of CPU when a program crashes
- Logic analysis cannot be carried if the µp has built in cache and execute instructions out of it.

2.4.2.2 Testing the Hardware and Software along with the HOST using In Circuit Emulators

A micro controller which is part of an embedded system is replaced by an In circuit Emulator in the Target Machine. The Emulation Software is stored in a separate memory different from application memory and the emulation software maintains Execution status of the Application in its memory. The execution of the Application is controlled by the emulation software. The emulation software also will have communication software component to communicate with the HOST. HOST can initiate series of test cases to In-Circuit Emulator and the test execution results are sent back to the HOST. Fig 2.14 shows the interfacing of target machine with the HOST Machine.
The emulator software provides the support to test the Software along with the Hardware: Emulator software supports debugging of the application through commands sent from HOST. The following can be undertaken to facilitate the debugging of the Application.

- Allows to set break points
- Allows to examine the contents of Memory and Registers
- Allows to see source code
- Stops execution and resume execution
- Allows single stepping through the code
- Allows to see the contents of the memory and registers even when the application is crashed
- Emulators captures trace of program execution
- In the event of any failures, Emulators still help in interacting with it through Host machine. The entire dump of overlay memory can be viewed and the reason for failure can be investigated.
- The emulator software can be read from a host and copied into overlay memory
Different types of Testing can be done using the In-Circuit Emulator

- Testing the inbuilt Peripheral devices
- Testing for the memory Leakages
- Testing for the response time of a function
- Testing for the function by function usage analysis
- Testing for the code which is not used by the Application
- Testing for changes in data at specified at memory Locations
- Testing for high used functions
- Testing for inter task communication through Mail boxes, Queues and Pipes including the overflow and underflow conditions

2.4.2.3 Testing the Hardware and Software Along with the HOST using Software Monitors

A monitor is software that has two components each residing on the HOST and the TRAGET. The Components communicate with each other. The Component on the target provides for Testing and debugging facilities by way of interacting with the embedded application.

The interfacing of Target with HOST through Monitors is Fig 2.15.

- Monitor resides on the ROM
- Has serial or network interface to read the application and store in the RAM and run it
- Can set the Break Points, examine and set memory and registers

The Monitors that runs on the host does the following.

- Provides communication Interface with monitor on the target
- Provides debugging Interface
Monitor on the host can communicate with monitor on the Target to place the code in RAM or flash. If necessary some of the locator functions are executed in the process.

The monitor on the target can do the following.

- Users can interact with the monitor on the host to set break points, run the program and the results are communicated to monitor on target for facilitating the execution.

Monitors can be used to carry the following types of testing:

- Testing for the memory Leakages
- Testing for the response time of a function
- Testing for the function by function usage analysis
- Testing for Week code which is not used by the Application
• Testing for changes in data at specified memory Locations
• Testing for high used functions
• Testing for inter task communication through Mail boxes, Queues and Pipes including the overflow and underflow conditions

2.4.2.4 Test case scenarios for testing Hardware and Software Along with the HOST

The above testing mechanisms can be used to undertake the following types of testing using Target, HOST and the Test Gadgets:

1. **Testing for the shared data problems**
   
   • To test the occurrence of shared data problem
   • Testing for the occurrence of Interrupt Priority inversion
   • Testing for the occurrence of dead Locks

2. **Testing throughput and response**
   
   • To test the response time for each of the Event.
   • To test Response time when several of the events occur simultaneously
   • To test the number of Events processed per second

3. **Testing for portability**
   
   • Testing for Endean
   • Testing for Variable conversion
   • Testing for Parity
   • Testing for several Addressing Modes
4. Testing Built-In peripherals

- Test code related to inbuilt DMA
- Test code related to UART
- Test code related to ROM
- Test Code related to Cache

5. Testing Power Saving Requirements

- To switch off the power to micro processor and to test switching on the power to Micro Processor when any of the events Occurs
- To test the powering on and off of the Peripherals through Software
- Manually Turning on the power of the Microprocessor and to check whether the Micro processor is executing the application from the location from where it is suspended

6. User interface testing

- When key board inputs are involved, to test whether event processing is done the way it is desired
- When an LCD display is required whether output is displayed properly as desired

2.5 Test case requirements and testing Types

Various types of testing are to be conducted to ensure that the embedded systems are built properly. The testing types include unit testing, Environment testing, Integration Testing, END-TO-END Testing and Regression testing.

Testing requirements can be projected right in the beginning of the development of the system. The testing scenarios can be projected from the
end user point of view. Test scenarios involve conducting any of the testing types.

Unit testing of Hardware devices or software components/functions are to be conducted to confirm that the Hardware units and software units are developed properly. Environment testing is to be conducted that the Hardware environment is properly set for the software to run properly.

Integration testing is to be conducted to test that the Hardware devices are connected properly and the software units are function with the desired interfaces specifically related to argument passing and returning of the values. The Integration of Hardware with software must also be tested especially when hardware devices initiate interrupts and the interrupt service routines are invoked properly and the code in the interrupt routines are handling the devices properly.

Embedded Systems are event driven and the events initiated because of the stimulus caused by the production system. Events are processed and the control action is initiated as a consequence of the events. The events are to be processed as per some patterns which actually impose prior and post conditions especially with reference to timings. The proper processing of the events must be tested by conducting END-TO-END Testing. END-TO-END Testing is a kind of Integration testing but conducted from the end user point of View. It gives the minimal of testing required to ensure that the events are being processed.

Embedded systems shall undergo changes either due to change in Hardware or both. When changes are made, the same are to be tested quite rapidly. The change made at one location may affect some other locations of Hardware, or software or both. The changes must be regressed over Hardware and Software and all affected components are to be tested. It is
necessary to select all the test cases that relate to changes made and the effected components.

The type of testing to be done requires a decision on the location at which the testing is to be carried and also the kind of test method that should be used. The test cases will be generated to carry testing at location using a test method conforming to the type of testing required. Fig 2.16 explains the connectivity test cases that should be considered for carrying the testing.

**Fig 2.16 The Testing Connectivity**

### 2.6 The Test Process Architecture

The testing architecture that encompasses the process of undertaking the testing is shown in the fig 2.17.
The testing requirements projected above require an architectural framework as shown in the figure 2.17. The requirement of testing using HOST and TARGET needs two software agents on either side. All the test cases are stored centrally in a data base at HOST. The Database shall also have the scripts which define the schedule of conducting the testing. A central control component at HOST which is the ES test system submits the test cases to respective testing methods and the outcome of testing is stored in the database. ES test system shall have the interfaces to interact with simulators, Third Part tools, Soft Logic Analyzers, Assert Macro code generators and the Scaffolding code generators.
The soft Logic analyzers conduct the testing through probes and make available the test results to the ES system which in turn stores the test results in the database. The test cases however will be made available by ES test system to Soft Logic Analyzer.

The ES test system also makes available test cases to simulators which are given with the image as input. The simulators make available the outputs to the ES test system which stores the test results in the database.

The ES test system also makes available test cases to third party tools which are given with the image as input. The third party tools make available the outputs to the ES test system which stores the test results in the database.

The Es test system initiates Assert Macro Generators by making available a set of test cases. The Macro Generator changes the Application code, compiles links and relocates the code and executes the code. The output results are written into the database by the Macro Processors.

The Es test system initiates Assert Macro Generators by making available a set of test cases. The Macro Generator changes the Application code compiles links and relocates the code and then executes the code. The output results are written into the database by the Macro Processors.

The Es test system initiates Scaffolding Generator and processor by making available a set of test cases. The Macro Generator changes the Application code compiles links and relocates the code and then executes the code. The output results are written into the database by the Script processor Processors.

When testing is to be done using the In-circuit emulator or Monitor, the ES test system hands over the test cases to the agent on the target. The
agent on the Target hands over the test case to Emulators or Monitors as the case may be. The test results are made available by monitor or Emulator to test agent on the Target which in turn to Agent on the HOST. The agent then stores the test results into the database.

User interface on the HOST shall help conducting the audit trail and test coverage analysis.

2.7 Comprehensive testing requirements of an Embedded System

Comprehensive Testing of embedded systems requires the identification of various test methods, the location of carrying testing and the kind of testing that can be conducted using specific method. Comprehensive Testing includes testing Hardware, Software and both. The comprehensive testing includes various types of testing which includes unit, Integration; END-TO-END testing, Regression Testing and theses types of testing must be achieved through conducting different types of testing at different locations using different testing methods. Different types of testing’s that need to be carried at different location using different methods are summarized and place below.

2.7.1 Host Based testing requirements

2.7.1.1 Testing through Scaffolding

- Testing the hardware independent code simulating Input/output functions that deal with the Hardware devices
- Testing for processing input is fed by a single device through directly calling Interrupt service routines
- Testing passage of time through calling the interrupt service routine of the timer devices
- Testing for the Interaction between hardware devices
- Testing for interaction of Task control code with hardware devices
- Testing for interaction of Multiple tasks through control tasks to provide processing through the same device
- Testing Interaction between several devices
- Testing the chain of tasks by way of simulating the occurrence of a sequence of events
- Testing the chain of tasks by simulating the simultaneous occurrence of a set of events
- Testing the tasks by simulating the occurrence of uncommon events
- Testing the Tasks for the failures of communication interfaces
- Testing the interaction of tasks with control task that deal with the devices by way of bypassing the interrupts initiated by the Hardware
- Testing the process that several tasks initiates requests to be processed by a device and the device initiates an interrupt that processing is complete.
- Testing repeated requests initiated by external interrupts for processing by the same device specially controlling the time delay between different types of processing
- Testing the communication between the multiple instances of hardware independent code
- Testing for the proper operation of resources such as Queues, Mail Boxes and Pipes
- Testing for the overflow condition i.e. trying to write to the queue or mail box or pipe when it is already fully written
- Testing for underflow condition i.e. trying to read a queue, or Mail Box or Pipe when it is empty

2.7.1.2 Testing through simulator Interface

- Response time for each of the event based processing
• Response time for processing the events that occur simultaneously.
• Testing for Portability for each of the cases below:
  o Significance of the Bytes (Endean)
  o Processing 16 Bit and 32 Bit variables in an expression
  o Type Casting
  o Addressing
  o Parity
  o Floating point processing

• Testing the Built-in peripheral devices
  o Reading from ROM
  o Reading from RAM and writing into RAM
  o Peripheral Interface Testing through DMA
• Throughput based testing
• Should test the startup code
  o To test whether the requisite tasks are identified with the Real time operating system
  o To Test whether memory pools are created properly
  o To test whether the handles required related to queues, Mail Boxes, Pipes, events and Timing have been created within the startup code
• Should test Assembly code
• Should support script based testing
• Testing for the shared data problems
  o To test the occurrence of shared data problem
  o Testing for the occurrence of Interrupt-Priority inversion
  o Testing for the occurrence of Dead Locks
2.7.1.3 Testing through Assert Macros

- Testing for Usage of proper Devices addresses specially MAC addresses.
- Checking for proper data bits related to the devices
- Checking for proper setting of the value
- Checking for Range of values to be contained in a Variable
- Checking for proper Function Calling Sequences

2.7.1.4 Testing through Third Party Tools

**Memory based testing**

- Memory Leaks due to the call to OS
- Memory Leaks due to the calls to third party tools
- Testing for writing to the end of an Array
- Testing for usage of freed pointers
- Testing for proper Memory allocation without any overlapping as the system runs
- Writing to a Queue when it is full
- Reading from a queue when it is empty
- Record of memory allocation
- Graphical display of memory allocation and usage

**Profile based Testing**

- Testing functional interaction on a complete tree of functions
- Testing for Task Interactions
- Testing for the High Use routines
- Testing for stack allocation for every call
- Record Usage Histories
- Testing for race condition
- Testing for priority Inversion
o Testing for deadly embrace
o Testing for passing of parameters due to function calls
o Testing for multithreaded processing of functions

**Variable based Testing**

o Test for change of value of a Variable or set of variables and also recording of peak values of the values
o Test for change of values of the variables by triggering certain events
o Test for variables by name when in scope, the stack, and registers of the target processor

**Testing for coverage of code**

o Testing for coverage of a functions
o Testing for coverage of a block of code
o Testing for coverage of Control Structures
o Testing for converge of Loop Structures

2.7.2 Testing through Target and HOST

2.7.2.1 Testing through Logic Analysis

**Testing in timing Mode**

- Timing of the signals
- Sequence of occurrence of signals
- Pattern of occurrence of signals
- Validity of the Signals
Testing in state mode

- To test whether proper addresses are fetched for instructions
- To check the kind of values being written to the memory addresses
- The state of signal when a clock input is either edge triggered or level triggered in terms of address in the address lines, data in the data lines, Read/Write on the control lines and Read enable/Write Enable Signals
- Storing the trace of instructions executed
- To check the time at which the first instruction of an interrupt routine executed after asserting a Interrupt signal by peripheral devices.
- To trace the uncommon events occurring through tracing and storing the signals
- To test the time durations during which a particular device is accesses by way of defining filters.
- To test the sequence of occurrence of a signals along with mapping of execution of the Embedded Systems Code to trace the areas of problem occurrence

2.7.2.2 Testing through Emulators

- Testing the inbuilt Peripheral devices
- Testing for the memory Leakages
- Testing for the response time of a function
- Testing for the function by function usage analysis
- Testing for Week code which is not used by the Application
- Testing for changes in data at specified at memory Locations
- Testing for high used functions
- Testing for inter task communication through Mail boxes, Queues and Pipes including the overflow and underflow conditions
2.7.2.3 Testing power management of the devices

- Testing Power Saving Requirements.
- To switch off the power to micro processor and to test switching on the power to Micro Processor when any of the events Occurs
- To test the powering on and off of the Peripherals through Software
- Manually turning on the power of the Microprocessor and to check whether the Micro processor is executing the application from location from where it is suspended

2.7.2.4 User interface testing

- When key board inputs are involved, to test whether event processing is done the way it is desired
- When an LCD display is required whether output is displayed properly as desired

2.7.2.5 Testing for JTAG based debugging

- Testing for scanning of Multiple JTAG Device connections
- Testing for multiple debugging connections to JTAG Devices

2.7.2.6 Testing the functioning of the devices

- Testing for Hardware System bring up
- Testing for Faulty devices early in the development life cycle
- Testing of processor activity of bus devices
2.8 Conclusions

In this Chapter, the need for testing the embedded systems comprehensively has been presented. Types of test cases that must be considered for testing the embedded systems comprehensively have been presented.

The classification of test case types required for testing at HOST and at HOST along with Target has been presented. Various Methods are proposed for testing the embedded systems which include scaffolding, Assert Macros, Simulators, Third party tools, in-circuit emulators and monitors. The Test case types that must be used with respect to each of the Test methods have been presented.