Automation of Combinatorial Interaction Test (CIT) Case Generation and Execution for Requirements based Testing (RBT) of Complex Avionics Systems

P Venkata Sarla[1]
Research Scholar, Bharathiar University,Coimbatore
Scientist-G, Aeronautical Development Agency,
Bangalore, India

Dr. Balakrishnan Ramadoss[2]
Professor, Department of Computer Applications,
National Institute of Technology,
Trichy, India

***Abstract*—In the field of avionics, most of the software systems are either safety critical or mission critical. These systems are developed with high quality standards strictly following the relevant guidelines and procedures. Due to the high criticality of the systems, it is mandatory that the verification and validation of these systems are done with utmost importance and only then any system is cleared for flight trials. The verification and validation activities need to be very exhaustive and hence take a considerable amount of time in the software development lifecycle. This paper describes about the innovative approach towards automation of Combinatorial Interaction Test case generation and execution for Requirements Based Testing of complex avionics systems for achieving test adequacy in a highly time efficient and cost efficient manner.**

***Keywords*—Avionics; combinatorial interaction testing; requirement specifications; requirements based testing; safety critical; validation; verification**

I. Introduction

Avionics systems are complex real time embedded systems with a very high criticality associated with them. These systems are software intensive and exhaustive verification and validation activities need to be carried out both at system level and software level to ensure error free and safe functioning of the system. Verification of the Software Development Life Cycle (SDLC) deliverables right from requirements engineering phase is essential in order to ensure that defects are discovered early and fixed as doing it at later stages has high impact on cost and effort.

The validation testing of avionics system is done with the Software Under Test (SUT) running on the actual target hardware and all the interfacing subsystems simulated. Implementation of each of the functionality is tested by running a number of test cases on the SUT. The test cases for the Functionality Under Test (FUT) are designed to uncover errors, demonstrate that the inputs are properly accepted by the SUT and the outputs are correctly produced. Validation testing is basically black box testing that examines the aspects of system functionality with little regard for the internal logical structure of the software. The SUT and the simulated systems run in real time during the validation tests.

A. Combinatoral Interaction Testing

Combinatorial Interaction Testing (CIT) can detect failures triggered by interactions of parameters in the SUT with a covering array test suite which tests all the required parameter value combinations. Traditionally testers develop scenarios of how an application will be used, then select inputs that will exercise each of the application features using representative values, normally supplemented with extreme values to test the performance and reliability. The problem with this often ad hoc approach is that unusual combinations will usually be missed, so that a system may pass all tests and may work well under normal circumstances, but may eventually encounter a combination of inputs that it fails to process correctly. By testing all combinations, for a specific interaction strength within the input variables, CIT can help to avoid this type of situation.

B. Requirements based Testing

A general principle of good requirements engineering practice [1] is that requirements should be testable. Requirements Based Testing (RBT), therefore, is a systematic approach to test case design where you consider each requirement and derive a set of tests for it. RBT is done to demonstrate that the system has properly implemented its requirements. By combining methods from requirements engineering and software testing, this testing methodology provides a set of quality assurance activities and management tools that enable getting requirements right from the outset. The RBT process addresses two major issues [2] first, validating that the requirements are correct, complete, unambiguous, and logically consistent; and second, designing a necessary and sufficient (from a black box perspective) set of test cases from those requirements, to ensure that the design and code fully meet the requirements. When designing tests, two issues need to be overcome: reducing the enormous number of potential tests down a reasonable size set and ensuring that the tests got the right answer for the right reason. The RBT process will drive out ambiguity and drive down the level of detail. The overall RBT strategy is to integrate testing throughout the SDLC and focus on the quality of the requirements specification. This leads to early defect detection

which has been shown to be much less expensive than finding defects during integration testing or later. The RBT process also has a focus on defect prevention, not just defect detection. The test cases for each of the FUT are designed using the corresponding Software Requirements Specifications (SRS) and Interface Requirement Specifications (IRS). For the FUT, the requirements related to processing of input data and generation of output data are specified in the SRS. The address and format of the input and output parameters are defined in the IRS between the SUT and the interfacing subsystems for the FUT.

C. Contents of the Pape

The rest of the paper is structured as follows. Section II discusses literature survey on related work. Section III introduces the new approach of CIT for RBT of complex avionics systems which is explained with a case study detailed in sections IV and V. In Section IV the manual testing approach used for a Mission critical system in a combat aircraft/helicopter is explained followed by the disadvantages of manual testing. In Section V automation of CIT cases generation and enhancement of the manual test rig for automatic execution for RBT of the system is elaborated followed by the advantages of automation.

II. Literature Survey

A. Automatic Test Data Generation

In [3] development of Test Case Generation (TCG) algorithm for CIT and idea for considering input constraints and building a unit testing harness from TCG is addressed. In [4] and [5], the authors have used programs from Software-artifact Infrastructure Repository (SIR) as their subjects for examining the effectiveness of CIT on regression testing. In [6], the authors illustrated that adding constraints in CIT of highly configurable systems, reduces the number of feasible system configurations but it is not guaranteed to reduce the size of the CIT sample to achieve coverage of desired strength. In [7] covers discussion on integrated approach for finding covering arrays and application of the same for constructing variable strength arrays. In [8] an approach to automate unit and integrating testing of radio's control software is described. In [9], the authors have illustrated an automated approach for finding and fixing conformance faults between given software system and its combinatorial model. In [10] automatic generation of test configurations that cover all pair-wise interactions using feature models for testing Software Product Line (SPL) is explained. In [11] the authors have proposed a framework for automated pair-wise testing of SPL, with an objective to generate the minimal set of test configurations that are valid and cover all pair-wise feature interactions.

B. MC/DC Coverage with CIT

In [12] automatic test data generation for testing of C programs at white box level for obtaining multiple coverage criteria including MCDC is covered. In [13], the authors have discussed about the extent of statement/branch and MC/DC coverage and the Fault Detection Rate (FDR) that can be achieved by executing CIT cases with strength varying from 2 to 5 on two subjects taken from SIR. They have taken original implementation and number of mutants of the subjects to study the effectiveness of CIT. Though they have generated covering arrays with strength of 5, as executing all the test cases with higher combination strength is a huge effort, the number of test cases that were executed is limited to that with strength 4. But a decrease in the statement/branch/MCDC coverage and FDR was noted by running only a subset of test cases with higher strength when compared to execution of complete set of test cases with lower strength.

But for mission critical and safety critical avionics software systems, it is essential to achieve 100% statement/branch and MC/DC coverage so that the FDR also is high. In f DO-178B guidelines followed for development and certification of avionics systems, RBT is emphasized because this testing strategy is found to be most effective in revealing errors. From the new approach followed by us we have found that the advantages of CIT for RBT are more compared to CIT with higher strength than needed.

III. Introduction to New Approach

We have evolved a new approach of performing CIT for RBT for verification and validation of complex avionics systems involving interactions of varying strengths within the parameters of the functionalities. CIT for RBT with minimum required strength is more effective in uncovering the errors with lesser effort than performing CIT with higher strength than required. For computing the expected output of the CIT cases for each FUT, the corresponding reference models are developed using the corresponding SRS and IRS. Because of this approach, the requirements get refined at initial stages of SDLC saving time of rework if detected later. The required optimal strength for CIT of the FUT is derived from the requirements thus validated and elaborated instead of generating and executing CIT cases with higher strength than needed. This approach which provides the benefits of both CIT and RBT involves the following activities:

1) Generation of expected output for each test case

As the FUTs will be computation intensive involving number of parameters, for generating expected output for each set of inputs, reference models are developed.

2) Generation of optimal, reusable combinatorial interaction test cases for RBT

Because of complex nature of requirements and the typical constraints on values of input as well as intermediate and output parameters for the systems of this domain, for generation of test data, additional considerations are required as compared to systems of other domains. Hence this activity is automated by enhancing the reference models for the FUTs and integrating with covering array generation tool for CIT suite.

3) Execution of CIT for RBT

As the systems are highly interface intensive with a number of other sub systems interfacing through various buses, feeding the inputs to the SUT for a particular FUT is highly cumbersome. Hence the test rig is enhanced and integrated with the CIT suite for automated execution of CIT for RBT.

4) Generation of test reports

As the number of test cases is too many, generation and tracking of test results manually is very difficult. Hence the test report generation activity is automated.

As per [15], generally, automation always follows manual testing. Typically, one or more rounds of manual testing already would be performed on the automated test rig. This implies that manual test cases already exist and have been executed at least once. But in our approach, even the first time execution of test cases can be done automatically.

IV. Case Study

To explain automation of test data generation for CIT and automation of execution of the generated test cases, we have taken the case of Mission Management Computer (MMC) software as a case study.

Introduction to the MMC which is the SUT, manual test rig used for black box testing, manual test procedure and the drawbacks of the same are explained in sections A to D below. Further in section V the automation of all the activities for the new approach of CIT for RBT of MMC is explained.

A. Mission Management Compute

In a modern combat aircraft/helicopter, the Mission Management Computer (MMC) is a highly complex unified software system. It is the heart of the avionics architecture which is the bus controller for more than 25 subsystems connected to it namely, Weapon Management System, Multi Mode Radar, Laser Designation Pod, Cockpit Controls System Redundancy management system, Data Acquisition System, Fuel System, Engine System, Electrical System, Brake Management System, Hydraulics System, Environment Control System, Recording And Replay Systems, multiple display processors for displaying more than 1000 symbols on MFD (Multi Function Display unit), Head-Up Display (HUD) unit, HMD (Helmet Mounted Display), Communication Systems, Backup Instruments, Vehicle Health Management Systems, Flight Control System. These systems interface with MMC on different buses like 1553B, RS422, Video and discretes.

Most of the mission management functions of the combat aircraft are implemented in MMC, mainly, weapon management for various modes of air-to-ground and air-to-air attack functions, redundancy management of the external interface systems for fault tolerance, Pilot Vehicle Interface (PVI) functions that include processing of cockpit controls, driving various display surfaces and Warning/Caution management, sensor management functions and so on. Thus there are thousands of software requirements for the MMC system to receive and process data from multiple subsystems and transmit the processed data to other subsystems. The MMC software is developed incrementally with a set of new functionalities added during each iteration.

B. Manual Test Rig for Testing of MMC

1) Components of the Manual Test Rig

In order to facilitate testing of various functionalities implemented in the SUT, the test facility has the following components as shown in Fig–1.

- Desk top PCs for Simulated Interface Models (SIMs) of the subsystems interfacing with the MMC on different data buses same as in the target aircraft.
- Provision of connecting MMC (SUT).
- Power Supply unit for SUT and the SIMs.
- Patch panel for feeding /tapping various signals/data.

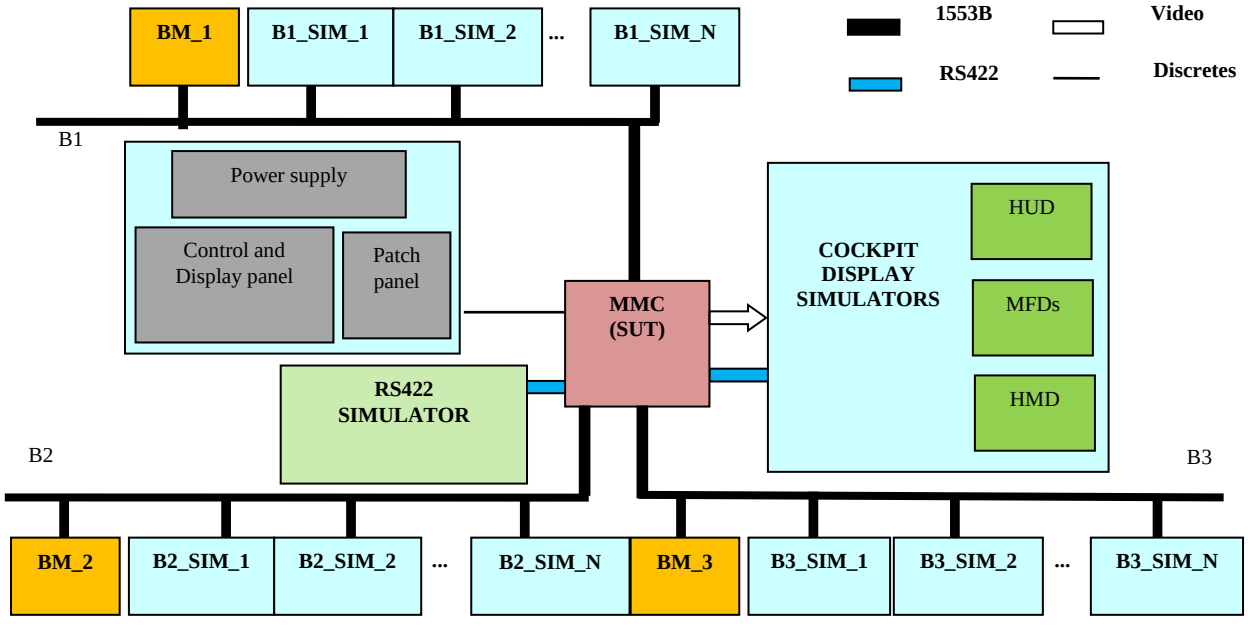

Fig. 1. Test Rig for Manual Testing of MMC.

- Control and display panel for ON/OFF control of SUT, isolation of SUT from bus, monitoring the health status of various components of the test rig etc.
- Bus Monitors (BM_1/2/3) for capturing data on the 1553B buses for analysis.
- RS422 Simulator for simulating RS422 interfaces to SUT same as in the target vehicle.
- Cockpit display simulators – Head Up Display (HUD), Multi-Function Displays (MFDs) and Helmet Mounted Display (HMD).

2) Simulated Interface Models

The test rig which is used for black box testing of MMC comprises of a network of Simulated Interface Models (SIMs) of various subsystems as depicted in Fig-1. The SIMs mimic the actual subsystems connected to the SUT in the target vehicle in terms of inputs and outputs to the SUT.

Each SIM will have the provision for feeding the inputs through GUI in engineering formats in a Transmit (TX) Window. These values are converted by the SIM into the required digital format and are updated on the corresponding data bus. Similarly, the outputs from the SUT which will be in digital format (Hexadecimal/Binary/Octal/ASCII/BCD are received by the SIM to which the data is addressed. They will be translated into engineering format and displayed in its GUI in Receive (RX) Window.

The UI of the SIMs shall also have the provision to feed transmit data in digital format which will get automatically updated in engineering format in the TX window. Similarly, there shall be provision to view the received data in digital format also. The test cases are static in the sense, for a particular test case, the values fed from the SIMs are constant. Hence the expected output of the SUT for the test case will be constant. However, the simulated subsystems and the SUT run dynamically in real time as per the bus scheduler functionality.

C. Manual Procedure for Generation and Execution of Test Cases for MMC

For each of the functionalities implemented in the MMC, test cases are generated and executed manually as follows:

1) The input parameters for the FUT are identified.

2) The address of the interfacing sub-system for each of the input parameters, the corresponding message details (the bits /words) and the range of values of the input parameters, are identified from the IRS.

3) Test cases are generated to ensure that the SUT is tested for different values of each of the input parameters and different combinations of input parameters. The values chosen for the input parameters include boundary values and mid value of the range. Additionally, as per the SRS, if there are any decisions/conditions with respect to specific values of any parameter, then values >, < and = to the specific value are added.

4) For each of the test case, the expected output value is calculated based on the SRS and converted into the format as per the IRS and specified in the test case document.

5) For running a test case on the test rig, the tester needs to feed the input values in various SIMs as per the test case, observe the output on the cockpit display surfaces and/or on the RX window of the SIMs which consume the output of the SUT corresponding to the FUT.

6) The observed output is compared with the expected output manually and the test result as PASS or FAIL is updated in the test report.

D. Drawbacks of Manual Methods of Generation and Execution of Test Cases

The process of generation of test cases manually is not very efficient and has many drawbacks. The test cases are not easily retrievable and reusable for regression testing of incremental software upgrades. The extent of combination coverage and path coverage depends on the randomly selected values of the input parameter.

The manual execution of test cases is time consuming, cumbersome and non repeatable as explained below.

- Values of different input parameters for the test case need to be provided across different subsystem terminals manually.
- If a sequence of inputs is needed to be provided within a specific timeframe consecutively, it is very difficult to achieve in the current approach. Requires multiple retries.
- Output result needs to be observed across multiple subsystem terminals manually.
- If the requirement is to update the output data only for a specific duration (for e.g., setting of a FLAG for one cycle), it is very difficult to observe the same. Tester needs to capture the data using the bus monitor terminal during run time and analyse offline whether the output data is updated correctly during the expected time duration in correlation with the input data.
- If any test case fails, then for demonstrating the failure to the designers, the whole process needs to be done again manually. If any observation is non-repeatable, getting the right scenario to get the observation becomes impossible some times.
- The systems are developed as unified systems suitable for different variants of target vehicle. For similar functionality, the expected behaviour of the system for the same input conditions will be different across different modes of operation (for e.g., navigation, approach, landing, weapon aiming, weapon releasing, exiting from attack mode, jettisoning etc.) and for different variants of the target vehicle (Airforce/Navy /Fighter/Trainer). The test cases are not easily reusable. A particular testing scenario if needs to be repeated for a different mode, then all the set of inputs need to be provided manually again across multiple terminals.
- Even if there are minor changes in the upgraded software releases, regression testing for clearance of

the upgraded version of the software takes same time as taken for the initial clearance of the software.

- Though modular design methodologies are used, every time when the software is upgraded for new functionalities, the impact of the changes on the existing software is very huge. The reason is the complex nature and huge size of the order of three-to-four million lines of source code. Performing detailed impact analysis manually is impractical. Even the usage of Computer Aided Software Engineering (CASE) tools for impact analysis of this type of software upgrades on the previously working software has been proven to be impractical. The tool shows hundreds of relationships across various objects of the source code. So based on gross level impact analysis carried out manually and with the knowledge of previous defect history most of the testing is repeated for previously implemented requirements. But as explained above, the effort for re-executing the test cases is very huge.

Because of above reasons, though the effort involved in automation is significant, it is one time effort which will help in reducing the regression testing time drastically for various upgrades of MMC software. During the development and maintenance of avionics systems which extend to about 15-20 years, there will be hundreds of software upgrades released incrementally. Once the setup for generation and execution of CIT for such systems is established, the same can be reused with no or minimum changes for testing the upgraded versions during each iteration. The test case generation and execution activities become more of process dependent than person dependent. In this domain where attrition of test engineers is very high, having this type of process dependent testing mechanism helps in a very big way for the organization.

V. Automation of CIT for RBT of MMC

In order to increase the efficiency of testing by using CIT and to ensure 100% requirements coverage for the RBT of MMC, the following activities are automated.

a) Generation of Expected Output for each test case by developing reference models for the FUT

b) Generation of Test data for the CIT cases

c) Execution of Combinatorial Interaction Test Cases for RBT of MMC

d) Generation of Test Report

Out of the above four activities that are automated, (b) & (c) explained in section B and C are unique to MMC testing. These methods are first time evolved and applied and are highly beneficial in many ways as explained in section VI. Though (a) and (d) are similar for systems belonging to various domains, (a) is explained in section A in brief as the same is used as the basis for (b). (d) is covered in section D for completeness.

A. Automation of Generation of Expected Output

Development of the reference models as shown in Fig-2 for different FUTs is carried out for generating the expected output values for every test case. The reference models are independently developed by the testing team based on the corresponding SRS. Generally the programming language used for the reference model is different from the one used by the design team in the actual SUT. The values for the input parameters for each test case are based on the IRS between the SUT and the interfacing subsystems for the FUT.

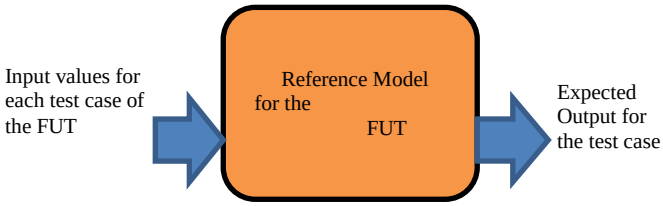

Fig. 2. Use of Reference Models for Generation of Expected Output.

B. Automation of Test Case Generation

Automation of generation of test data for the test cases of CIT involves enhancement of the reference model for the FUTs to include Constraint Checker (CC) and integration of the same with covering array generation tool. The test cases with the input test data and the expected output data are stored in the CIT Suite. For CIT to be effective for avionics systems, additional considerations are necessary for generation of optimal CIT suite with respect to CIT of highly configurable systems.

1) Additional steps to be taken for generation of test data for effective CIT of avionics systems

The steps for evolving the required combinations that need to be covered for evolving the test data for CIT of MMC software are depicted in Fig-3. The reasons for these additional considerations are:

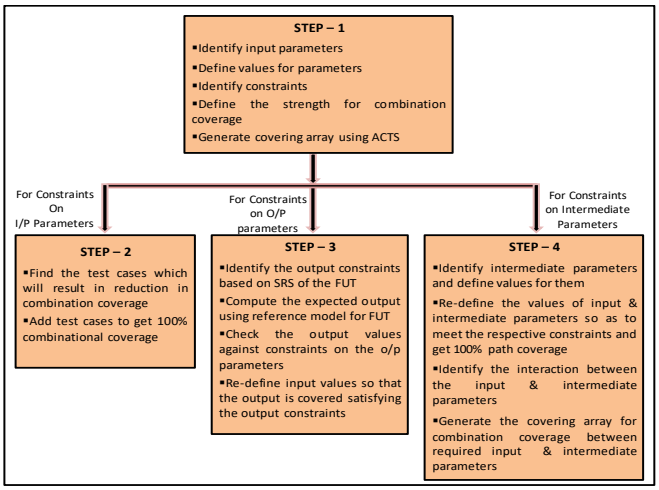

Fig. 3. Steps to be followed for Generating Combinatorial Test Cases for MMC.

a) Need for additional test cases to meet typical constraints on input parameters.

Most of the input parameters to MMC are associated with validity bits. As per the requirements whenever validity bit of any parameter is received as INVALID, MMC uses the previous valid value for a specified duration. Because of this type of constraints on the input parameters, the SUT will not

get tested for some combinations of values of inputs. Hence CIT is an effective test generation technique for avionics systems only when additional test cases are added to meet the complex nature of input constraints and to get 100% combination coverage of required strength.

b) Need for redefinition of values of input parameters to meet typical constraints on intermediate parameters and output parameters.

Many of the functionalities that are implemented in the MMC software are computation intensive. The computed values are used either to display a symbol and/or data on the cockpit display surfaces and/or send the data to other systems for consumption. For a set of inputs, if the output which is the current_location of a moving symbol on a cockpit display surface is out of the dispay_area or Field Of View (FOV) then, based on the requirements, EITHER the symbol shall be made absent OR the symbol shall be displayed at the boundary in flashing.

Because of the reasons mentioned above, the choice of values of the input parameters should be such that the corresponding output values do not result in the location of the symbol being out of FOV. Only two test cases are required to test the symbol for absence/flashing. All other test cases should be such that resultant locations of symbol are distributed across the entire FOV instead of cluttered at some portions.

Similarly if the computed data is sent to other equipment, the values of the input parameters in the test cases should be such that the resulting output values are within valid output range, the values are distributed across the entire range of the output parameter and the Output_Data INVALIDITY bit is not set for more than one test case.

Every combination in the test suite needs to be checked by running on the corresponding simulated reference model (refer Fig-2) to ensure whether the resultant values of the output parameters are meeting the above output constraints. If not, the input values need to be redefined. Thus there is an impact of output constraints on the selection of values for various input parameters.

Similarly, in the algorithms for various functionalities, there will be constraints on the intermediate variables which are dependent on input variables. Based on the values of these intermediate variables, the program takes multiple paths. In order to ensure that the test cases are adequate enough to cover all the paths, it is essential that these types of constraints on intermediate values are met. Accordingly the values of the input parameters need to be redefined.

For effective CIT, wherever there are constraints on the values of intermediate/output parameters, the values of corresponding input parameters need to be redefined to meet those constraints. However, the size of the test suite and the combinations should not increase significantly.

c) Need for generation of covering arrays with combination coverage for the input parameters and the intermediate parameters.

The existing combination strategies [16] [17] are inadequate for handling intermediate parameters for combination coverage required for avionics software testing. In the algorithms for different functionalities, there will be decisions/paths based on conditions with combinations of input and intermediate parameters. Hence generation of covering array with combinations of input parameters alone will not be adequate. The covering array needs to be generated with combination coverage for the input parameters and the intermediate parameters to get 100% condition/decision coverage.

2) Development of facility for evolving CIT cases for RBT of MMC

We have developed the Combinatorial Interaction Test case Evolving Facility (CITEF). This facility is useful for evolving optimal test cases with input values for the parameters of the FUT such that the typical constraints on the input/intermediate and output parameters of the FUT in MMC are met. Fig-4 shows the Block Diagram of CITEF. It comprises of Input Value Generator (IVG), Reference Model (RM) of the FUT and CTCG tool for covering array generation. The RM in turn has two components: The Simulated Functionality Under Test (SFUT) and Constraint Checker (CC). SFUT is developed independently by test team members based on the corresponding requirements specified in the SRS of MMC. CC shall have the intermediate and the output constraints applicable for the FUT stored in it. The IVG is GUI based application which has the provision for entering the initial set of input values and range of data for each parameter.

The initial set of input values are derived through category partitioning [18] which involves selection of typical representative values based on input domain partitioning and boundary values as per the interface requirements. The IVG has the provision to manually update the values of the parameters or to automatically select random values (without duplication) from the given range of the parameters. This provision is given so that the size of the test suite does not increase abnormally. Else IVG was selecting random values from the valid range and with an increase in number of values of any parameter, the size of the test suite increases exponentially and the testing time will proportionally increase.

We have used ACTS Version 3.1 released in April 2018 for generating covering arrays for CIT of MMC. ACTS [19] [20] [14] is a GUI-based CIT tool developed by National Institute of Standards and Technology (NIST). ACTS has the ability to generate tests with interaction strength from 2-way to 6-way, with a user-friendly GUI and a command line version suitable for use in scripts or system calls from another tool.

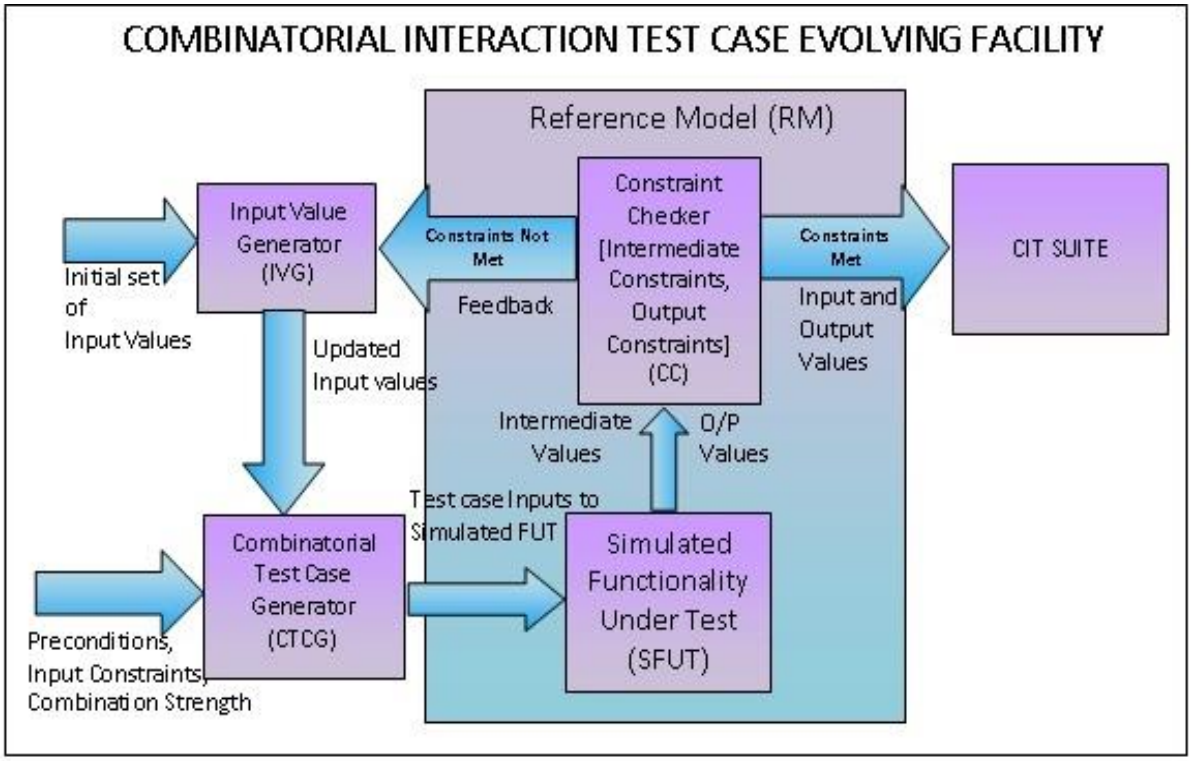

Fig. 4. Combinatorial Interaction Test Case Evolving Facility (CITEF).

Combination Test Case Generator (CTCG) developed by us is an application to which ACTS tool is integrated. The values supplied by the IVG are accepted by an input file within the CTCG. These values along with the constraints on the input parameters and the combinations coverage strength (mixed strength wherever required) are fed to the ACTs tool. The generated test cases are stored in an output file. For the FUT, each of the generated combinatorial test cases shall be run on the SFUT in order. During execution of each test case, the run time intermediate values and the output values are sent from the SFUT to the CC in the RM. These values are checked by the CC against the respective constraints which are stored in it. If they are not satisfied, then feedback from CC is sent IVG and execution of further test cases on SFUT is stopped. The feedback information will contain the test case number, values of the intermediate and output parameters and the information about constraints that are not met. The same is displayed in the GUI of IVG. On selecting the EDIT option on the IVG for a particular input parameter, the values of that parameter will change randomly or a fixed value can be fed by the user. When the new values are applied by pressing the APPLY button, the same are sent to the CCTG Tool for generation of updated covering array. Each of the newly generated combinatorial test cases shall be again run on the SFUT. This process is repeated till the optimal values are assigned to the input parameters for every test case with values of all the input parameters satisfying the constraints on intermediate parameters and output parameters. The final test suite shall be such that on running all the test cases on the MMC, 100% path coverage and combination coverage of the parameters shall be achieved.

The test cases with the generated test data are stored in the Test Case Library in the CIT SUITE along with the preconditions and the expected output value for each test case. For each of the input and output parameters the corresponding address details (BUS ID, SIM ID, MESSAGE ID, Word number and Bit details) are also stored. The test data can be automatically fed to the SUT at black box level through the test rig as explained in the following section.

C. Automation of Execution of Test Cases

For Automation of execution of RBT of MMC, the test rig used for manual testing (Fig-1) is augmented and further integrated with CIT SUITE of CITEF as depicted in Fig-5.

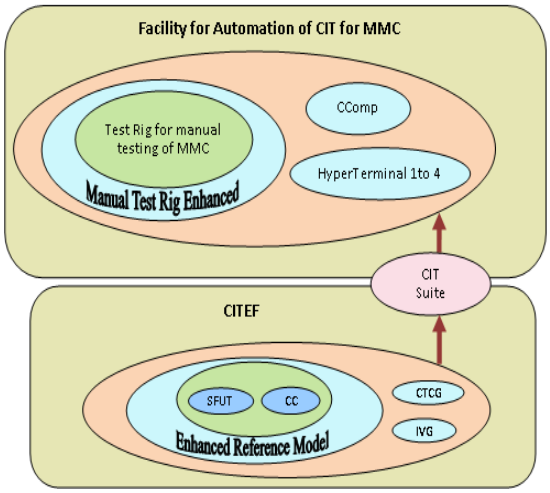

Fig. 5. Depiction of Components of Facility for Automation of CIT for RBT.

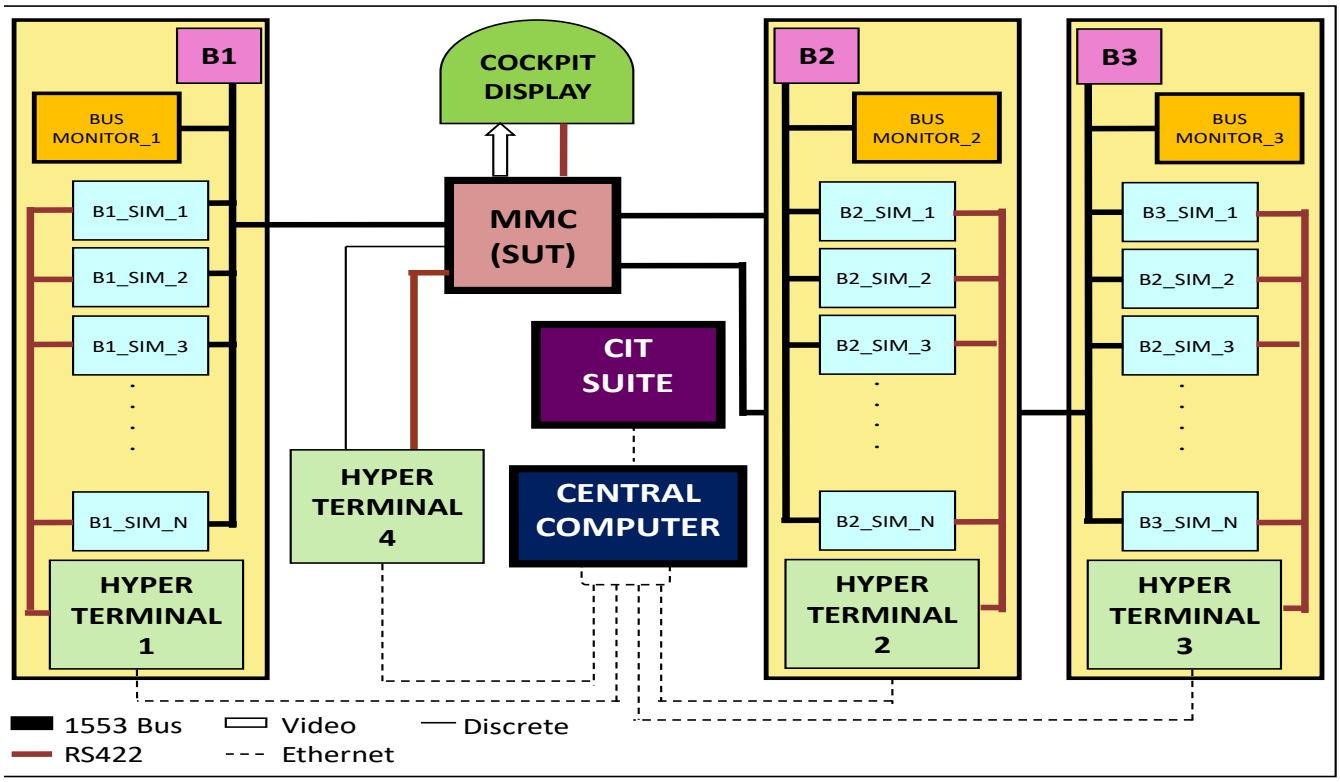

Fig. 6. Test Rig for Automation of CIT for RBT of MMC.

The block diagram of the Test rig for automation of execution of CIT for RBT of MMC is depicted in Fig-6.

Each SIM has a unique identification number BX_SIMY where X and Y are variables. X is the Bus 1/2/3 on which the SIM is connected. Y is the SIM number on that bus. For e.g.

Sub-System_1 on Bus1: B1_SIM1

Sub-System_2 on Bus 1: B1_SIM2

The test cases for the FUT are selected from the Test Case Library in the CIT SUITE. The Central Computer (CCOM) is the interface between the CIT SUITE and the Test Rig. For a chosen test case, the sources (respective SIMs) for input parameters are identified. To each of the four hyper-terminals connected on the different buses, the information about the values that the SIMs need to update in specific messages on the respective buses is sent by the CCOM. Each of the hyper-terminals 1 to 3 in turn will send the address and data blocks to corresponding SIMs on the respective 1553B buses. The SIMs will put the data accordingly in their transmit buffers for updating on the bus. Hyper- terminal 4 will update the values on the RS422 bus and set the discrete values as required to be fed to the SUT as per the test case. The output of the SUT is observed on the cockpit display surfaces and /or the RX windows of the SIMs to which the data is addressed. In the Automated approach, all the SIMs and the SUT would be working coherently in the same way as during manual testing except that the input to the SUT from the SIMs are fed without human intervention. For e.g., in order to test the SUT for computation of MACH NUMBER DATA, the 'total pressure' and 'static pressure' values have to be provided through B1_SIM4 (Simulated Air Data Computer) and Aircraft_On_Gnd_In_Air information has to be provided through B2_SIM2 (Simulated Engine System Interface Unit). This happens automatically on selection of the relevant test case from the CIT Suite.

Further, the computed MACH NUMBER DATA for the inputs fed, can be seen in the RX Window of the GUI of B2_SIM4 (Simulated Flight Control Computer) and B3_SIM5 (Simulated Fuel System Interface Unit) and on the cockpit display surfaces.

Floating point numbers are not supported by ACTS. For floating point type of values, the tool was not considering the decimal portion of the given input values for usage in the constraints defined. This limitation of the tool also had to be handled in the test harness.

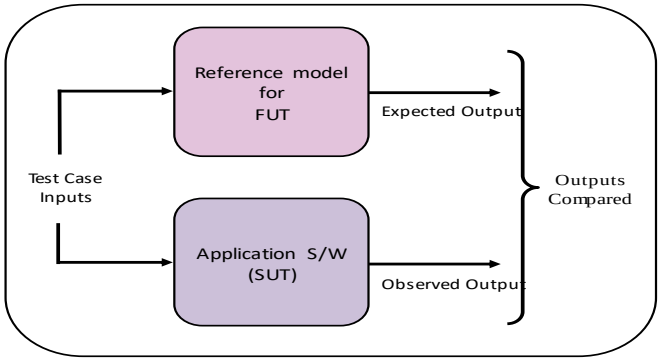

Fig. 7. Comparison of Expected and Observed Outputs for Test Report Generation.

D. Automation of Test Report Generation

The outputs from SUT captured by the bus monitors are compared with the expected output for automated test report generation. As shown in Fig-7, for a set of input values for a given test case if the corresponding outputs of the application software are same as that of the reference model, then the test is considered as PASS.

VI. Benefits of Automated Approach

- Tester need not switch to multiple terminals for providing inputs manually. Multiple Inputs from different SIMs will be provided automatically when test case is run from the CIT Suite.
- Sequence of inputs if required to be provided within a timeframe can be conveniently given as there is no human delay involved. The script has to be designed such that those input variables are placed and executed sequentially.
- The output that appears only for a short interval can also be easily verified as the checking of the output is automated.
- A particular testing scenario if need to be repeated, then it is sufficient to only re-run the test case instead of giving all the required inputs manually again.
- Regression testing for different variants of the target vehicle can be done at a faster pace and more efficiently.
- This process helps in effective CIT for RBT of MMC with typical constraints on values of inputs as well as intermediate and output parameters.
- The process of feeding the test data to the SUT automatically from the CIT SUITE and generation of Test report by comparing the output of the reference model and the actual output from the SUT reduces
- The overall testing time drastically as shown in the Table-1.

TABLE I. Comparison of Effort Involved in Manual Testing and Automated Testing

Functionality Under Test	***No. of Test cases***	***Time for Manual Execution (Mins)***	***Time for automated Execution (Mins)***
Jettisoning of selected stores	14	70	7
MARK & UPDATE Functions with different types of sensors	45	240	20
Attack Functions in different guided modes	56	560	56
Send Specific Data	60	180	10

VII. Conclusions

As the procedure involves generation of test cases by development of the simulated reference model for the functionality under test based on the detailed SRS, any ambiguity in the SRS can be reported to the authors for correction/elaboration. This helps in detailing the software requirement specifications without any ambiguity which is the main goal of RBT and hence the combinatorial test cases designed using this method will generate the most optimal test cases. Execution of these test cases shall not only provide the benefits of CIT but also provide benefits of RBT.

Automation is beneficial only if the components of the automated test rig and CITEF: SIMs, CCOM, Hyper-terminals, SFUT, CC, IVG etc. are validated for correctness. Errors in any of these components may result in the following which are not acceptable.

- FALSE NEGATIVE errors: Not detecting the errors present in the SUT which is very dangerous as errors in mission critical and safety critical avionics systems when encountered during flight can even lead to catastrophic consequences.
- FALSE POSITIVE errors: Though the implementation in the SUT is correct, this is highly undesirable as it results in waste of time in analysing and tracing the reason for the failure to a bug in the test facility.

Floating point numbers are not supported by ACTS. For floating point type of values, the tool was not considering the decimal portion of the given input values for usage in the constraints defined. This limitation of the tool also had to be handled in the test harness.

VIII. Scope for Future Work

Floating point numbers in constraints are not supported by covering array generation tools. There is scope for further work in development of tools for handling floating point data type. The automation test facility can be enhanced for optimisation of test cases in which multiple Functionalities can be tested together instead of sequential execution.

Acknowledgment

We wish to acknowledge our gratitude to the management of Aeronautical Development Agency (ADA), Bangalore for permitting publication of this paper.

References

[1] Sommerville Ian. "Software engineering", Pearson Education, Inc., publishing as Addison-Wesley, 2009.

[2] Predrag Skokovic, Marija Rakic-Skokovic, "Requirements - based testing process in practice", IJIEM International journal of industrial engineering and management, 2010, Vol. 1, p. 155-161.

[3] Yu-Wen Tung, , Wafa S Aldiwan, "Automating test case generation for the new generation mission software system" DOI 10.1109/Aero.2000.879426, 2001.

[4] Xiao Qu, Myra B Cohen, Katherine M Woolf., "Combinatorial interaction regression testing: A study of test case generation and prioritization", IEEE, 2007.

[5] Manuj Aggarwal, and Sangeeta Sabharwal, "Prioritization techniques in combinatorial testing : A survey", 1st India International Conference on Infromation Processing IICIP, 2016.

[6] Myra B Cohen, Matthew B Dwyer and Jiangfan Shi, "Interaction testing of highly configurable systems in the presence of constraints", ISSTA , 2007.

[7] Myra B Cohen and Charles J Colbourn, "Constructing test suites for interaction testing", IEEE 25th International conference on software engineering ICSE 03, 2003.

[8] Redge Bartholomew, "An industry proof-of-concept demonstration of automated combinatorial test", IEEE, 2013 p. 118 to124.

[9] Angelo Gargantini, Justyna Petke, and Marco Radavelli, "Combinatorial interaction testing for automated constraint repair", 10th IEEE International conference on software testing, verification and validation workshops, 2017.

[10] Aymeric Hervieu, Benoit Baudry and Arnaud Gotlieb, "Pacogen: Automatic Generation of pairwise test configurations from feature models". Proceedings of international symposium on software reliability engineering (ISSRE'11) Nov 2011, Hiroshima, Japan. 2011. <hal-00699558>.

[11] Dusica Marijan, Arnaud Gotlieb, Sagar Sen and Aymeric Hervieu, "Practical pairwise tesing for software product lines" SPLC 2013, Tokyo, Japan 2013 <hal-00859438>.

[12] Prasad Bokil, Priyanka Darke and Ulka Shrotri, "Automatic test data generation for C programs", Third IEEE international conference on secure softwawe integration and reliability improvement. 2009

[13] Dong Li, Linghuan Hu, Ruizhi Gao, W Eric Wong, D Richard Kuhn, and Raghu N Kacker, "Improving MC/DC and fault detection strength using combinatorial testing", IEEE International conference on software quality, reliability and security, 2017, p 297 to 303.

[14] Mehra N Borazjany, Linbin Yu, Yu Lei, Raghu Kacker and, Rick Kuhn, "Combinatorial testing of ACTS : A case study", IEEE Fifth international conference on software testing, verification and validation , 2012, p. 591 to 600

[15] https://www.softwaretestinghelp.com. Practical software testing, June 2018.

[16] Mats Grindal, Jeff Offutt, and Sten F Andler, "Combination testing strategies : A survey", GMU Technical report ISE-TR-04-05, July 2004.

[17] Mats Grindal, Bitgitta Lindstrom, Jeff Offut and Sten F, Andler, "An Evaluation of combination strategies for test case selection" GMU Technical report, 2006-10-06.

[18] Sunint Kaur Khasla and Yvan Labiche, "An extension of category partition testing for highly constrained systems", IEEE 17th International symposium on high assurance systems engineering, 2016 p. 47 to 54.

[19] Bestoun S Ahmed, Kamal Z Zamli, Wasif Afzal, and Miroslav Bures , "Constrained interaction testing: A systematic literature study", 2017, IEEE Access, DOI 10.1109/Access 2017.2771562 Vol. 5, p. 25706 to 25730.

[20] Sunint Kaur Khalsa and Yvan Labiche,"An Orchestrated survey of available algorithms and tools for combinatorial testing", Research Gate, 2014.