AN INTEGRATED MUTATION ANALYSIS TOOL (IMAT) FOR SIMULINK MODELS AND THEIR IDEAL IMPLEMENTATION

Manju Nanda, J. Jayanthi, V. Madhan, Sameer Nayak
Aerospace Electronics Division
CSIR-National Aerospace Laboratories (NAL)
Kodihalli Post, HAL Airport Road
Bangalore-560 017, India
Email : manjun@nal.res.in

Abstract

Mutation analysis is proving to be an effective approach in detecting the design flaws for safety system designs such as flight control systems or the stall warning systems. The drawback of these analyses is that they are time-consuming and difficult-to-use. This paper discusses about the mutation tool, IMAT, developed in-house for detecting the design flaws of safety critical systems designed in SIMULINK. The IMAT tool is developed as a plug-in to the SIMULINK tool suite and enhances the performance of the model-based development of safety critical systems. Some of the unique features of the tool are ease-of-use, portable, scalable, user-friendly and easy to analyse the reports.

Keywords: Mutation Testing, Mutation Score, MATLAB/Simulink/Stateflow Models, Mutation Operators

Introduction

Model based design, development and qualification approach is being widely used across the industries for implementing a safe, reliable, available, complete and correct system as per the project specifications. Aerospace domain has also embraced model-based engineering process; Simulink tool suite is one of the widely used model-based tool to design and analyze control applications. Mutation analysis of the Simulink models provide an effective way of capturing the design flaws in the model and generating efficient safety test-cases for testing the design at the model level and detecting design flaws early in the engineering process.

Mutation is a method to introduce small syntactic changes in the model that can cause an error in the system; this changed model is called a mutant model. The change is deliberately introduced to compare the response of the original model with the mutant. The comparison is evaluated in the mutation analysis has been improved the mutation analysis by reducing unnecessary mutants generated, adding temporal logic verification technique and eliminating redundant test cases. By introducing these features, time required for analysis is reduced as number of mutants is reduced and the temporal logics also verified.

This paper discusses the mutation tool, IMAT, developed in-house to perform the mutation analysis of the SIMULINK models. The tool not only automates the mutation process but also integrate to Simulink as a plug-in. IMAT can be executed by the users by a single click. This performance enhancement is a non-invasive approach. Some of the novel and unique feature of the tool is its intelligence in parsing the SIMULINK designs and providing the information about the probable mutants that can be generated. The mutation tool activates the SIMULINK Design Verifier (SlDV) to generate the safety test cases required to check the system function after improving the design based on the feedback by the IMAT tool.

The modified improved workflow by introduction of the IMAT with the Simulink improves the model-based design and development approach towards automation and safety. The tool develops mutation for logical, mathematical, gain, constant, relational, sign, sum, min-max, trigonometric and state flow blocks used in designing the
safety critical systems. After correcting the flaws, SIDV checks the mutant design for functionality by generating the counter examples. The combination of IMAT-SIDV [3] provides an effective set of test cases for testing complex designs at model level.

**Improved Model Design Workflow for Safety Critical Systems**

A safety critical system must be designed to avoid failures in most complex functionalities. Yet, if failures prevail one of the major reasons would be inadequacy in testing. In conventional approach, this could be due to lack of understanding of the design, tool limitation, human errors, inappropriate modelling, or improper translation from design to model. Testing of complex systems manually needs persons with very good experience and expertise in the specific domain. There is a need to make the process formal and less error prone [1] [2]. Automation is the solution for making this process effective. To overcome these issues in the manual testing approach, an automated mutation analysis has been used as a formal approach to verify system models and generate test cases automatically. In our work SIDV formally verifies the Simulink models using the model checker technique. IMAT, the tool developed can be integrated with SIDV [3]. The Mathworks (Simulink, SIDV [3]) tool was considered for this approach as it is widely used in designing of control and monitoring systems for safety critical applications. IMAT, reduces the human error in the conventional approach [2] [7]. The tool has the intelligence to select of mutation operators automatically. Fig. 1 shows the improved model based design workflow for safety critical systems where IMAT is introduced in the model-based workflow. In this workflow we generate a set of test cases, which is a combination of SIDV and mutations. This integration provides an effective set of test cases detecting the design faults early in the engineering process, thus improving the design process of the model-based engineering process.

The improved workflow is implemented on a proven safety critical system: Stall warning/ Aircraft interface computer. Earlier work by us [19] proved the effectiveness of integrating model-checker with the mutation analysis. This integration of mutation with the SIDV is a new approach we propose in improving the model-based engineering process for safety critical systems.

**Integration of Mutation and SIDV**

The primary idea of integrating the mutation analysis with the SIDV is to automate the process of model verification. The SIDV generates test sets automatically, whereas the mutation analysis evaluates the effectiveness of each test case. With this integration there is an increase in the number of test cases generated automatically, improving the functional coverage of the model. The improved method also covers some of the critical functionalities of the model under analysis, as the test sets are generated for all the mutants. For a mutant whose functionality is different than the original model, the model checker generates a different test set. This test set has some common test cases as well as some unique test vectors. These unique test vectors are used to identify the mutants and analyse if the critical functionalities are simulated.

In the integration approach, the Model Under Test (MUT) is subjected to mutation with user selected mutation operators. These operators mutate the original model to create all possible mutants as per the mutation operators selected. Each model is then subjected to model checking using SIDV to generate a set of test cases for each mutant and the original model. The common/ same test sets are eliminated to reduce the analysis time. These test sets are used to simulate mutants; the results are compared with the simulated results for the original model. In case the results do not match, the mutant is killed else the mutant is still alive. The mutants could be alive/ unidentified due to the presence of an equivalent mutant or the test sets generated which are not effective enough to identify all the mutants. The analysis of equivalent mutant is done by the designer. The integration of mutation analysis with formal method based technique for generating an effective set of test cases in the verification of design process was carried out as a case study and is discussed in [19]. To realize this approach for model-based engineering process, integration of SIDV with a mutation tool is required.

**Improved Technique**

The work done and discussed in [19] has been improved by modifying the mutation operators and test cases. The mutation operators have been modified to eliminate the possibility of generating a model which is the same as the MUT in the form of a mutant i.e.; the earlier mutation operators generated some mutants which are exact replicas of the original model test set.
Related Work

We briefly relate our work to other tools that generate test vectors by means of software model checkers. These implementations are very similar to ours. Harman provides a list of mutation test tools. The MUT, which increased the computation time without improving the test set and hence, operators were modified to eliminate such possibility. The test cases generated were only for a specific length of time and could not verify the temporal logic decisions. Therefore, a provision to modify the time length of the test cases is introduced for the temporal logic verification. With this improvement time related logic can be verified. The SlDV is applied on every mutant generated by mutation tool. Many test cases were repeated as most of the objectives are similar for two different mutants of the same model. To avoid this repetition the test case sorting technique collected a set of unique test cases that could be used to simulate the models and complete the analysis.

To analyse the effectiveness of the test case, the ratio of number of mutants killed by the tests to total number of mutants generated (by the mutation operator) is evaluated, and the ratio is referred as mutant score. The higher the mutant score of the test case more is its effectiveness in analysis. The number of mutants generated by the mutation operators is represented by $M$ and the number of mutants killed is $N$ and the equivalent mutants $E$ (number of equivalent mutants) the mutation score is defined as $[7, [14], [15]$. 

\[
\text{Mutation score} = \frac{N}{M - E}
\]

The common principle underlying Mutation Testing [26] work is that the faults used by Mutation Testing [26] represent the mistakes that programmers often make. By carefully choosing the location and type of mutant, we can also simulate any test adequacy criteria.

This paper describes the related work which describes the existing mutation testing tools, the overview of the IMAT tool, the brief introduction about stall warning systems, the comparison results and equality check, the uniqueness of the tool and summarizes our tool findings on the mutation testing [26] applied to MATLAB/Simulink/Stateflow model and proposal for future work.

Rothermel [22] propose use of mutations to prioritise test cases to increase a test suite’s rate of fault detection. Schuler et al. [23] discuss the impact of equivalent mutations (mutations that keep the semantics of the model unchanged) and present an approach to detect such mutations by means of checking dynamic invariants. Most of the mutation tools are being used at the code level. Just et al. [24] describe the mutation tool, MAJOR for the Java compiler which reduces the mutant generation time and enables efficient mutation analysis. Hafiz [25] describes the mutation testing tool that provides the mutation plug-in with Eclipse framework. Jham et al. [27] provide approach for various mutation tests for Java based applications.

For an effective verification integration of mutation analysis with model-checker is discussed by Fraser and Wotawa [7]. This approach helps in achieving the property coverage and generating an effective set of test cases.

SlDV is a supporting tool for Simulink toolset. It uses event B based model checker in verifying the model design. SlDV [3] uses formal methods to identify hard-to-find design errors in models without performing extensive tests or simulations. SlDV generates detailed test generation and property proving analysis reports. SlDV [3] checks for model compatibility as per the MAAB standards. Using SlDV [3] it is possible to specify the functional requirements of the model and verify them using property proving. SlDV [3] detects design errors such as integer overflow, dead logic, division by zero and violations of design properties and assertions. It generates test cases automatically as per the design specifications and objectives. The generated test cases known as counter examples provide simulation inputs that exercise model functionality. SlDV [3] mathematically proves whether those properties are satisfied and, if not, provides counterexamples that would violate the properties. As a result, developers can find design flaws, unsatisfied requirements, and unreachable states or logic that would be difficult to uncover using simulation alone.

SlDV [3] uses "Polyspace" and "Prover Plug-In" formal analysis engines to analyse the model and capture the functional requirements and model coverage [9]. The test vector generated from the analysis includes condition, decision, and modified condition/decision (MCDC). The SlDV [3] begins with the initial configuration of the model and can span an arbitrary number of time steps of analysis [9]. Ideally after the analysis, the results are equivalent to those of a complete search of every possible infinite sequence of input parameters. Block reduction is a Simulink
feature that is used to achieve faster execution during model simulation and in generated code. When the SlDV [3] translates a model, block reduction happens automatically, and blocks in unused code paths are eliminated from the model. The SlDV [3] results do not include test objectives for blocks that have been reduced [9].

The back-end also links the mutation analysis to SlDV to generate the complete test scenarios for large and complex system test cases are generated by simplifying the analysis using mathematical techniques [9]. But due to various factors and tool limitations the test vectors generated are not very efficient and results in lower model coverage. Hence mutations are expected to introduce test vectors to fill in the missing coverage. With this approach, models can be tested and verified automatically to achieve maximum functional coverage and ensure safety at the design phase. Fig.2 illustrates the screenshot of test cases generated by SlDV [3]. The modeling analysis capability is an effective approach in model-based design and development. In our proposed approach we are proposing the integration of the mutation tool with Simulink Design Verifier [28].

Automating the integration of Mutation with Model Checker (SIDV) : Overview of the IMAT Tool

The need to automate the integration of mutation analysis with SlDV is done by means of tool, IMAT: Integrated Mutation Analysis Tool. This tool can be integrated to any tool that has a gateway for Simulink models. The IMAT tool has two major components: Graphic User Interface which acts as a front-end and interacts with the users: - Back-end parses the Simulink models and GUI architecture.

GUI Architecture

The GUI architecture of the IMAT is shown in Fig.3. The figure depicts the overall functionality of the tool.

The GUI was developed using the MATLAB script and developed as a plug-in to the MATLAB R2013a APPS. (Apps typically consist of a Graphical User Interface (GUI), underlying code for performing the intended actions, associated data, and other supporting files).

The IMAT GUI provides interface to five major tasks such as: parsing the Simulink model, generating the mutants for the model, mutation and SlDV analysis, and generation of the results.

IMAT is activated by the user with the help of a mutation button. Fig.4 shows the screenshot for starting the mutation application.

Selection of the Simulink models for parsing is shown in Fig.5. The selected model (.slx or .mdl) is then used for mutation analysis process.

Parsing the Selected Simulink Model for Mutant Generation

The parsing of the selected model is done by clicking on ‘Check models’. The parser checks the Simulink model for mutants that can be generated. These mutant operators are highlighted for the User so that if required the user can select all or specific operators for which the analysis needs to be carried out. Fig.6 shows the screenshot for selection of mutant operators in the IMAT.

Mutant Generation

The ‘Generate mutants’ tab shown in Fig.7a generates the mutants for MUT (Model Under Test).

Mutants are generated for each block in the model with respect to specific operator. The list of mutants created for these operators are saved in *.csv file (which can be read in Microsoft office Excel). This result is displayed on the MATLAB command window which can be used for future reference. The display on command window is shown in Fig.7b.

Integrating IMAT with SIDV

For integration of IMAT with SIDV, we need to set the following options in the ‘Settings’ option of SIDV: Ideal selection time: 100 (for small models), model coverage: MCDC, test case objectives: Combined Objectives. Fig.8a shows these settings for SIDV.

With the integration, SIDV verifies each of the mutants generated for the Simulink model. SIDV checks the model compatibility as per its requirements. It further checks the model for design errors such as; divided by zero, incomplete model, missing variable. In case of any error encountered, SIDV identifies the error and displays it. Fig.8b show the compatibility results.
Integrated Test Case Generation for the MUT

Figures 9a, 9b and 9c show the stages of the test case being generated by SLDV for the generated mutants. Fig.9a shows the status of loading test case duration, Fig.9b shows the start of the mutation analysis and Fig.9c shows the end of the mutation analysis. The mutation based test cases are generated in the form of signal builder as a part of the harness model. The test cases are saved in the *.mat files. Test cases generated for every model is collected and combined to for a complete test set where the redundant test cases are eliminated. Once these test cases are achieved, each mutant model is simulated and the response is compared with the response of the MUT. To keep track of which model is being simulated, the model name is displayed on the command window of MATLAB, as shown in the Fig.9a and Fig.9c.

Stall Warning/Aircraft Interface Computer System

The integration of IMAT with SLDV is implemented for a proven system: SWS/AIC. The performance of the integrated tool is shown by the number of test case generated with or without the mutants.

The Stall Warning/Aircraft Interface Computer System (SWS/AIC) is a safety critical system being designed and developed for the 14-seater civilian aircraft. The software is developed and certified as per RTCA DO-178B level A [16], [17] certification standards. The SWS/AIC system performs two major functionality: Stall computation and warning and computing and announcing various other warnings such as landing, takeoff, over speed, baromistamcth, hydraulic low pressure and pitch trim warning.

The stall computation and warning is a very critical functionality as it warns the pilot of the impeding stall. A failure to inform leads to loss of control of aircraft leading to loss of aircraft and life. Hence it is very critical to analyze the design of the stall computations. This system is already approved for flight test and hence it was used to prove the effectiveness of the IMAT tool. The stall computation and monitors was designed in Simulink and hence the integrated approach was implemented to generate metrics to quantify the effectiveness of this integrated, automated approach.

In this work, the first step was to identify the set of mutable blocks for which the operators were developed. These operators were developed as functions in M code (MATLAB script) to be used by the main program for mutation. Mutant generation was automated by the main mutation program which analysed the model to identify the various blocks present. Identification of these blocks were restricted by defining a search depth and the array of all the mutable blocks saved in a variable. Each of these blocks are then mutated one after the other and saved as a mutant model. Once the mutant set is generated, another program (in MATLAB script) operates the SLDV over each model in the mutant set and the original model. The input test set (generated by SLDV) is then sorted to eliminate the repeated test set and saved for simulation. The mutation analysis is performed by simulating each of the test cases on the original model and the mutant set, and the responses are compared [8], [15]. The list of mutants alive is saved as an array, so that the next test case is simulated only on the mutants that are not identified. This technique was used to save the simulation time. The work flow of this algorithm is graphically described in Fig.4. The landing gear, flap monitor, and squat subsystems were considered for complete analysis for basic results. The monitor subsystem is focused for analysis of a complex system. These systems are analysed individually and as a part of the main system to observe the effectiveness of this method and also to get more clear results. This system is already tested certified and is currently used in the SARAS aircraft. The test cases achieved are compared with the documented manual test set.

Comparison of Result and Equality Check: Performance Analysis

Figure 10, 11, 12 demonstrates the screenshots of comparison results in the generated mutants. Fig.10 demonstrates the list of changes generated in the mutation. Fig.11 illustrates the list of mutants killed in the mutation process. The generated test sets are used to simulate the set of mutants and the results are compared to the simulated results of the MUT. By this comparison it can be observed if the two models are different, the test cases that identify this difference is a valid test case and the identification is called a mutant kill.

Equivalent Mutants

Figure 12 elucidates the list of equivalent mutants in the mutation process. The diagram results the total number of equivalent mutants in the model (overspeed.slx). Fig.12 contains the lists of equivalent mutants, mutants killed,
mutation score and number of test cases generated. The integrated approach improves the performance metrics of the model-based engineering process. This can be seen by the test-cases generated for the SWS/AIC system functionality. Table-1 describes the number of test cases generated for some of the subsystems of SWS/AIC system.

As shown in the Table-1, the number of test cases with integrated approach is more as compared to test cases generated without mutants. There is an increase in the test cases which about 2 times or more providing behavioral coverage to the system design. The mutation score for this approach is shown in Table-2.

The Table-2 lists the test set generated for the subsystems by SIVD and is compared with the score achieved by integrated approach. This shows that the improved test set identifies most of the mutants generated and increases the coverage of the model. It has also been observed that the improvised test cases cover significant critical test cases which are crucial in the analysis of model. As a part of experimentation, the SLDV was implemented only on the mutants that were alive/unidentified by the test set generated for the original model. The analysis shows that, this technique does not cover the critical functionalities and also many of the mutants still remain unidentified. Hence to achieve the best coverage it is advised to implement SIVD on the entire mutation set.

### Table-1 : Number of Test Cases Generated for Each Subsystems

<table>
<thead>
<tr>
<th>Subsystem Name</th>
<th>Test Cases without Mutants (Using (SIDV))</th>
<th>Integrated Test Case (with Mutants)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overspeed</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>Landing Gear</td>
<td>7</td>
<td>21</td>
</tr>
<tr>
<td>Flap Position Monitor</td>
<td>10</td>
<td>66</td>
</tr>
<tr>
<td>Squat</td>
<td>6</td>
<td>40</td>
</tr>
<tr>
<td>Lgflapsquat Monitor</td>
<td>28</td>
<td>1795</td>
</tr>
<tr>
<td>Flags</td>
<td>46</td>
<td>1501</td>
</tr>
<tr>
<td>ILMinp</td>
<td>176</td>
<td>17363</td>
</tr>
<tr>
<td>ILMout</td>
<td>109</td>
<td>970</td>
</tr>
<tr>
<td>ILMstat</td>
<td>19</td>
<td>1360</td>
</tr>
</tbody>
</table>

### Table-2 : Mutations Score for Subsystems Analysed

<table>
<thead>
<tr>
<th>Subsystem Name</th>
<th>Mutation Score (Using SIDV)</th>
<th>Mutation Score (Using Mutation Analysis)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Landing Gear</td>
<td>0.8181</td>
<td>0.9090</td>
</tr>
<tr>
<td>Flap Position Monitor</td>
<td>0.7987</td>
<td>0.9512</td>
</tr>
<tr>
<td>Squat</td>
<td>0.8888</td>
<td>0.9600</td>
</tr>
<tr>
<td>Lgflapsquat Monitor</td>
<td>0.6530</td>
<td>0.8877</td>
</tr>
<tr>
<td>Flags</td>
<td>0.92</td>
<td>0.9558</td>
</tr>
<tr>
<td>ILMinp</td>
<td>0.5240</td>
<td>0.6880</td>
</tr>
<tr>
<td>ILMout</td>
<td>0.6402</td>
<td>0.8805</td>
</tr>
<tr>
<td>ILMstat</td>
<td>0.8852</td>
<td>0.8852</td>
</tr>
</tbody>
</table>

**Uniqueness of the Tool**

IMAT integration with SIVD provides integration of mutation with model-checker. This approach not only automates the process but also makes the model-based design analysis more effective. Some of the unique feature of the IMAT for the model-based design and development are:

- Simple and portable
- Mutant generation at model level
- Easy to handle and user friendly
- Direct plug-in with Simulink Design Verifier (SIDV)
- Mutant generation for logical operators, constant, state-flow, relational operators, gain, Min-Max, Sum, Sign, integrator and trigonometric
- Intelligence for selecting the mutation operators automatically

**Conclusion and Future Work**

This paper discusses about the mutation tool, IMAT, used in integration with model-based tool, Simulink in design analysis of safety critical systems at the model level. The tool architecture, features are discussed. The effectiveness of the integrated approach of mutation with model-checker to verify critical designs is demonstrated by applying for a proven system. The performance metrics of this approach is demonstrated by the mutation score and the effective test-cases generated.
The complete tool is easy to work as it provides complete automation to the design verifier. The tool contains minimal human involvement and significant performance improvement. This mutation testing tool executing the large number of mutants and also finds the equivalent mutants.

Future work includes integrating with coverage analysis tool to check the coverage of the test case, changing sequence of test cases to identify equivalent mutants due to dependency of previous outputs and different mutant generation technique such as cluster algorithm to define new mutation database.

Acknowledgements

The authors acknowledge the Director, CSIR-National Aerospace Laboratories, Bangalore for supporting this work.

References


3. www.mathworks.com


Fig.1 Improved Model Design Workflow
Fig. 2 Test Cases Generated by SIDV

Fig. 3 Execution Workflow of the IMAT Tool

Fig. 4 Starting of the Mutation Application
Fig. 8b Compatibility Checking for Generated Mutants of the Overspeed.slx

Fig. 9a Loading the Duration of Test Cases in the Form of Array

Fig. 9b Starting the Mutation Analysis for the Model Overspeed.slx

Fig. 9c End of Test Case Generation for the Model Overspeed.slx
Fig. 10 List of Changes in the Mutation (Overspeed_changes.csv)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Mutant1 Changed Constant in overspeed/Constant from 6 to 6.0317</td>
</tr>
<tr>
<td>3</td>
<td>Mutant2 Changed Constant in overspeed/Constant from 1 to 1.0034</td>
</tr>
<tr>
<td>4</td>
<td>Mutant3 Changed Constant in overspeed/Constant from 1 to 0.96184</td>
</tr>
<tr>
<td>5</td>
<td>Mutant4 Changed Logic in overspeed/Logical</td>
</tr>
<tr>
<td>6</td>
<td>Operator Operator from AND to XOR</td>
</tr>
<tr>
<td>7</td>
<td>Mutant5 Changed Logic in overspeed/Logical</td>
</tr>
<tr>
<td>8</td>
<td>Operator Operator from AND to OR</td>
</tr>
<tr>
<td>9</td>
<td>Mutant6 Changed Logic in overspeed/Logical</td>
</tr>
<tr>
<td>10</td>
<td>Operator Operator from AND to NAND</td>
</tr>
<tr>
<td>11</td>
<td>Mutant7 Changed Logic in overspeed/Logical</td>
</tr>
<tr>
<td>12</td>
<td>Operator Operator from AND to NOR</td>
</tr>
<tr>
<td>13</td>
<td>Mutant8 Changed Logic in overspeed/Logical</td>
</tr>
<tr>
<td>14</td>
<td>Operator1 Operator from AND to XOR</td>
</tr>
<tr>
<td>15</td>
<td>Mutant9 Changed Logic in overspeed/Logical</td>
</tr>
<tr>
<td>16</td>
<td>Operator1 Operator from AND to OR</td>
</tr>
<tr>
<td>17</td>
<td>Mutant10 Changed Logic in overspeed/Logical</td>
</tr>
<tr>
<td>18</td>
<td>Operator1 Operator from AND to NAND</td>
</tr>
<tr>
<td>19</td>
<td>Mutant11 Changed Logic in overspeed/Logical</td>
</tr>
<tr>
<td>20</td>
<td>Operator1 Operator from AND to NOR</td>
</tr>
</tbody>
</table>

Fig. 11 List of Mutants Killed

Fig. 12 List of Equivalent Mutants