Journal of Aeronautics & Aerospace Engineering
ISSN: 2168-9792

Rathina Kumar et al., J Aeronaut Aerospace Eng 2016, 5:2
DOI: 10.4172/2168-9792.1000167

Research Article Open Access

Analyze the Mode Transition Logic of Automatic Flight Control System using Semi-Formal Approach

Rathina Kumar V*, Nanda M and Jayanthi J

Department of Aerospace, Electronics and System Division, CSIR-National Aerospace Laboratories, Bangalore, India

Abstract

Autopilot system is a highly critical avionics system in modern aircraft as it steers the aircraft automatically. The autopilot is a highly complex system driven by a complex logic and is one of the major reasons for the accidents in automated airliner. The autopilot logic consists of the mode-transition logic which in automated mode steers the aircraft based on the aircraft aerodynamics. In the automated mode the correct and efficient working of the mode-transition is highly critical; hence a high assurance approach is required to analyze the logic for its functionality and performance.

In this paper, we present a semi-formal method based approach to analyze and validate the Mode-Transition Logic (MTL) for an indigenously developed commercial aircraft in the vertical and lateral directions. The MTL is analyzed and validated for its correct, complete, and reliable functionality and operation using Stateflow. The modeled MTL logic is validated for the allowed transitions based on the input combinations against the requirements for functionality and safety. The outcome of the approach shows encouraging results with respect to assurance in functionality, performance and safety in comparison to the conventional manual approach of testing. Similar semi-formal based approach can be used to reduce the design effort in the design and development of complex system designs as compared to the manual analysis.

Keywords: Autopilot; Mode transition logic; Semi-formal methods; Stateflow; Simulink design verifier; Model advisor; RTR; Reactis; Validator; Tester

Introduction

The complex system of Aircraft, such as Digital Autopilot system, Flight management system and Electronics map displays are used to reduce the workload of pilots and gave better information, improving the flight performance and improve situation awareness. In this avionics architecture the autopilot system have more add on compared to flight management system and it never integrated into a single system. The elements of current avionics system are distributed across the mode control panel, the control display unit and primary flight display which include flight mode annunciations. The autopilot has different types of modes such as heading select, heading hold, vertical speed hold. A mechanical, electrical and hydraulic device of the aircraft is used to assist the autopilot operation. The autopilot logic is used to guide an airplane with minimal or no assistance from the pilot [1].

The autopilot is designed by using complicated mode transition logics. The designer spend more time to design the mode transition logic and their safe transitions and the designs are more strengthened by using verification and validation techniques such as assertions, safe states and safe transitions. The incorrect mode transition logic has led to accidents in the past year. The accidents are overcome by improving the Mode transition logic analysis [2,3].

Mode confusion occurs when the pilot believes the current mode is different from actual mode but it's actually in correct mode instead of the correct mode pilot change inappropriate mode. Mode confusion can also occur when pilot does not understanding the behavior of mode transition logic and pilot has poor knowledge about the mode transition logic. Advancement in digital avionics system has accounted for much of the improvement in air safety seen over the last few decades. At same time the growing complexity places in the system and increase the risk of the mode transition logic. To fly commercial fly today, the pilots must be a master in several complex, dynamically interacting system and should know operating at different levels of automation [3].

In safety critical applications become higher safety and functionality assurance for using the formal method based techniques according to the civil aircraft standard RTCA DO178B [4]. This RTCA DO 178B standard is providing the guidelines for design, development, verification and validation of airborne software in safety critical system. A formal verification technique is a mathematically based languages tools and techniques to formally model the design based on the project requirements. These formal methods technique is used to analyze the system behavior for all the possibilities and converge of the system [5,6].

Model based design and development is used to create the model at each and every stage of the software lifecycle and automatic model transformation for example from code to model. It is well defined method and produces more sustainable software and it is give graphical notations and good abstracting details. This technique has been used in automated high speed train and car autopilot and now being used for aerospace domain for demonstrating the functional and safety properties [7,8].

Semi formal method is technique to analyze the system in model level and code level and then design the computerized version of it. The computerized system having the same structure and functions as we expected and it satisfied the requirements according the standard guidelines [5,6].

***Corresponding author:** Rathina Kumar V, Department of Aerospace, Electronics and System Division, CSIR-National Aerospace Laboratories, Bangalore, India, Tel: +91-80 25086019/20; E-mail: rathina2020@gmail.com

Received March 22, 2016; **Accepted** April 25, 2016; **Published** April 28, 2016

Citation: Rathina Kumar V, Nanda M, Jayanthi J (2016) Analyze the Mode Transition Logic of Automatic Flight Control System using Semi-Formal Approach. J Aeronaut Aerospace Eng 5: 167. doi:10.4172/2168-9792.1000167

The amount of software has increased significantly over the last years and therefore, the verification of embedded software has become of fundamental importance. The most commonly used approaches to verify embedded software are based on co-debugging or co-simulation, which have the coverage problem. Formal verification assures complete coverage but is limited to the size of module that can be verified. In this paper, we present a new semi-formal verification approach in order to verify temporal properties of embedded software, based on the combination of simulation and formal verification approaches. The semi-formal verification approach can be used to overcome the drawbacks of both dynamic and static verification. This approach combines the benefits of going deeper and covers exhaustively the state space of the system. The effectiveness of the semi-formal approach provides the foundation to use this approach in validating the functionality & performance of complex aircraft logic [5,6]. Earlier this logic was validated using conventional approach consisting of reviews, tests at code level, system level and aircraft level. This approach is not only very laborious but also has the drawback in detecting the logic flaws earlier in the phase [9-12].

MTL is a discrete event system with states, inputs, and outputs which is usually modeled as a finite state machine where the states represent the modes. MTL system receives inputs and based on the inputs and the current mode, transits to another mode and produces an output. The outputs are used to command the control surfaces of the aircraft appropriately. Each such computation is referred to as a software cycle [13-15]. The mode and their transitions in the current research have been represented in tabular forms. Mode Transition Logic (MTL) is a design module used by the flight director of an autopilot to switch modes of flight control. The possible transition values from a current value of a state variable are specified in the State Transition Matrix (STM). The actual transition of a state variable from the current value to a new value is allowed when the condition(s) given in the Condition Matrix (CM) are satisfied. Control modes are switched based on the events that are received as inputs or commands to the autopilot [16-18].

In this paper, we implement the MTL vertical and lateral modes using Simulink Stateflow for an indigenous civil aircraft. The modes and logic concerning lateral, vertical modes transition, mode possibilities in lateral and vertical modes are validated at the design level for functional, performance, and safety properties. Comparison of the proposed approach with the conventional approach shows the improvement in the process in understanding and validating the complex logic at the model level rather than at the code level

Literature Survey

The safety, functionality and performance validation and verification of the complex system has always been challenge. Complex logic, complex systems are validated using conventional techniques such as simulation, testing and reviews with bottom-up approach. With the technological evolution, the time from concept to certification of a system is reducing as the demand for these systems are increasing. For example the design to certification of Airbus A350 XBW was in a short span of time [19].

In safety critical domain such as automotive, railways, space and aeronautical the new technologies are complex performing multi-functions. For example car autopilot system, intelligent train control system, electrical flight control system and cyber rail [5,6]. To ensure that these systems perform the functionality as per the requirements of time and safety, lot of analysis is to be performed. Mathematical and graphical based approaches are proving to be more effective than textual based approach. Formal and Semi-formal methods are gaining popularity in validating complex system/ software logic.

Development of the autopilot, the array logic based technique has been used to reduce the design effort. It is easily understood and provides a very concise way of specifying a large number of transitions in simple tabular column. Vertical and lateral modes of the autopilot have been designed using mode transition logic and the technique has been extended to cover the navigation and approach modes [2,3]. All the possible mode transitions in the presence of external or internal event and performance criteria are presented in the subsequent sections. The mode and their transitions have been represented in tabular form called as array logic based technique.

One of the mode transition logic was analyzed by using formal method approach in the name of the paper is mode confusion analysis of a flight guidance system using formal methods and the main author is Anjali Joshi. In this paper they used NuSMV and theorem proving technique to analyze the mode confusion logic [6]. But here we are checking all the state of mode transition logic in semi formal approach using matlab simulink/stateflow tool suite.

In the modern autopilot systems having complex components that are detecting and avoiding collisions with other objects and allow aircraft to land in situation where a human cannot see the runway environment. In earlier days several accidents and incidents have been reported because of the autopilot failure. For example in 1983 the Korean Air Lines Flight 007 flying from Anchrage to Seoul. This aircraft deviated more than 200miles from it path (soviet territory) and got shot down killing the crew and the passengers. The aircraft accident analysis reported navigational failure as the cause of the air crash. It was found that the flight was initially in heading mode, later the pilot either forgot to select the inertial navigation system or otherwise the pilot might activated but system got never activated. The autopilot goes to inertial navigation mode when these following two conditions are satisfied. The aircraft path and the predefined path must be close and the distance between these two paths is within 7.5 miles. These two conditions are continuously checked by software it is called guard. The guard is a logical statement which is used for mode transition of aircraft. The mode is changed only the conditions are true [2-3]. In this case the mode could not be activated, which could be due to improper implementation of the MTL logic or pilot error or system error.

Lateral modes and logic concerning lateral mode transition are less complex compared to the prevailing methods for autopilot design [8]. Various mode possibilities of lateral mode transition in an autopilot is mentioned along with specification criteria's that bound these transition and these possible transitions were given a frame work using MATLAB software. System decomposition, abstraction, and distribution lead naturally to sub problems that can be addressed using formal methods and tools, such as mathematical modeling, control law synthesis, and control implementation verification. We classify these methods and tools, which rely heavily on mathematical formulations of the underlying problem.

Approach

The novel approach proposed works on the solid foundation of the conventional approach. The tabulated MTL and MTL simulation blocks are used as the reference for establishing the semi-formal approach. This is done to ensure the correct implementation of the semi-formal methodology for analyzing the complex MTL algorithm. Figure 1 show the technique which is followed. Simulink stateflow is used to implement the MTL logic. The output of the stateflow is compared with the expected output using conventional manual approach. The comparison proves the efficacy of this approach in terms of ease of implementation, ease of understanding, and the improvement in the process.

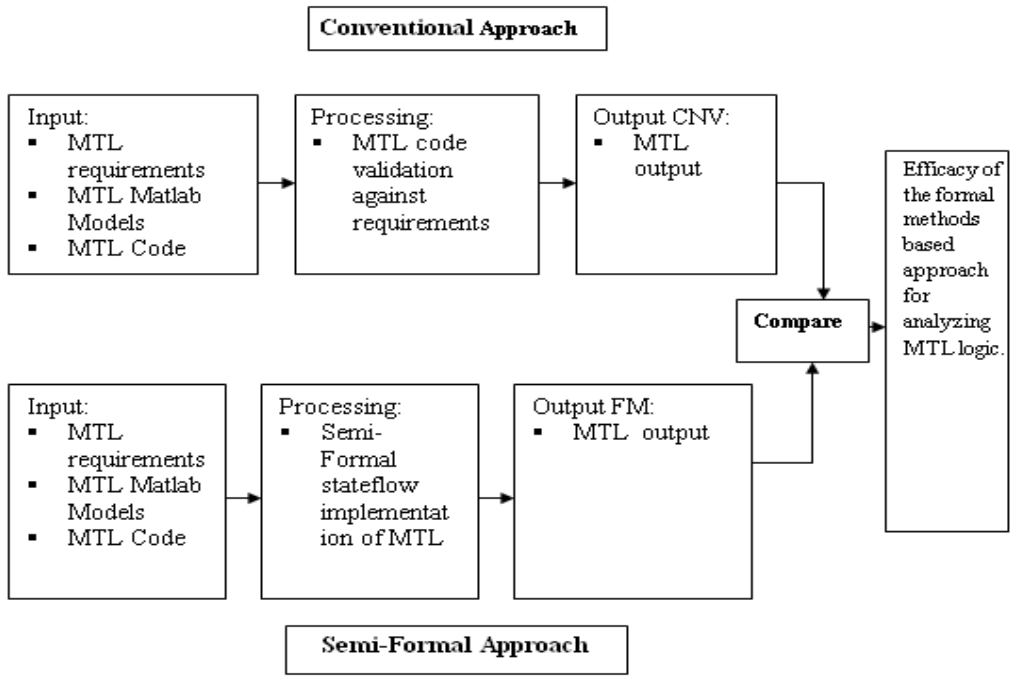


Figure 1: Approach used for this MTL analysis.

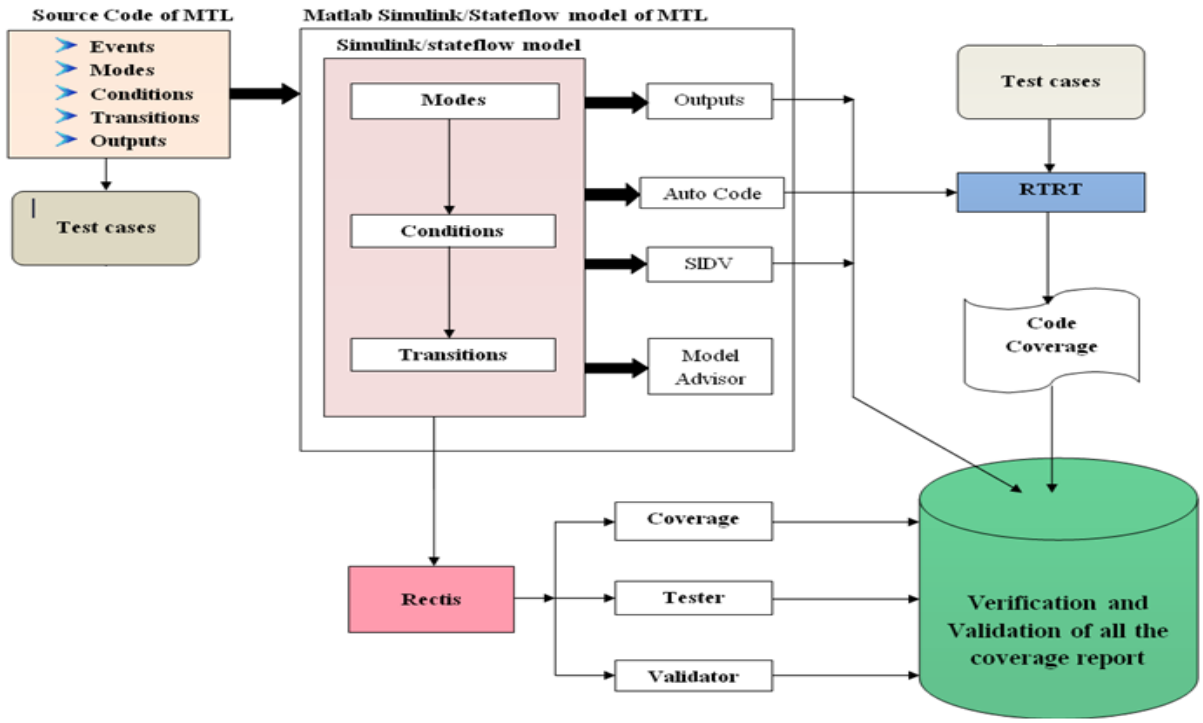


Figure 2: Flow of semi-formal method approach.

The inputs for the semi-formal method based approach are MTL requirements, MTL tables, and software code, as an input. These inputs are translated into the stateflow. The stateflow structure is similar to the code structure to maintain the semantic translation from code to model level. Stateflow standards are followed in order to generate compact auto code. Stateflow model is done according to the MAAB guidelines [20]. The equivalence checking of the MTL requirements is checked for the stateflow by mapping the MTL requirements to the stateflow.

To analyze the robustness of the stateflow design, test-scenarios

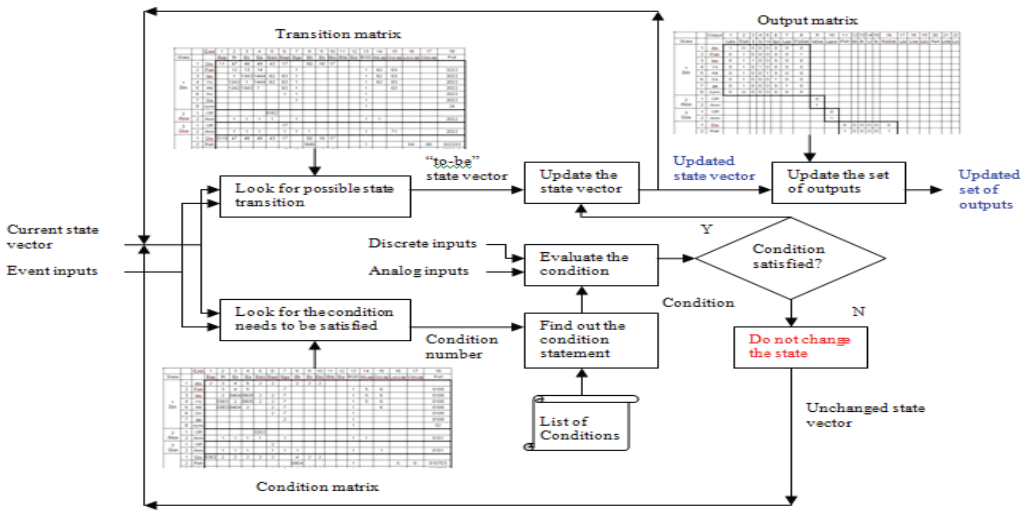


Figure 3: Software life cycle Logic of MTL [3].

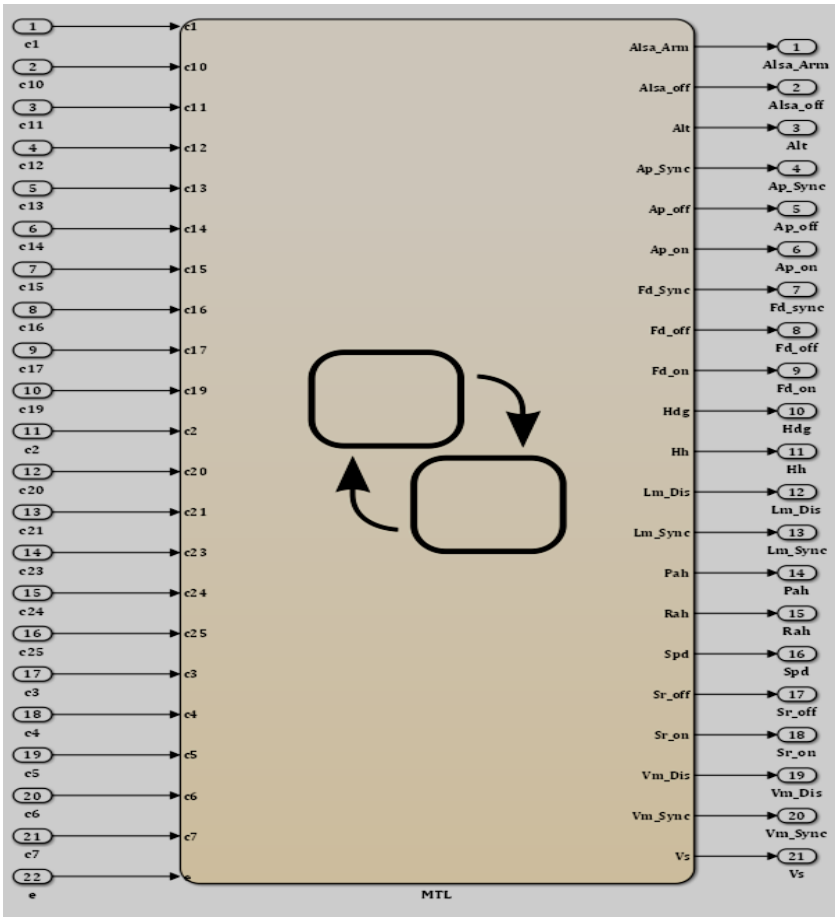


Figure 4: State flow model of MTL.

generated at code level are translated to model-level. The test outputs at the model level are compared to the code level test output. This comparison provides the correct semantic translation of the code to model.

Figure 2 shows the details of the implementation of the approach for the MTL logic. The MTL model using the stateflow is generated based on the manual code structure and the MTL tables for inputs, outputs, events, conditions and transitions. Model inputs are events

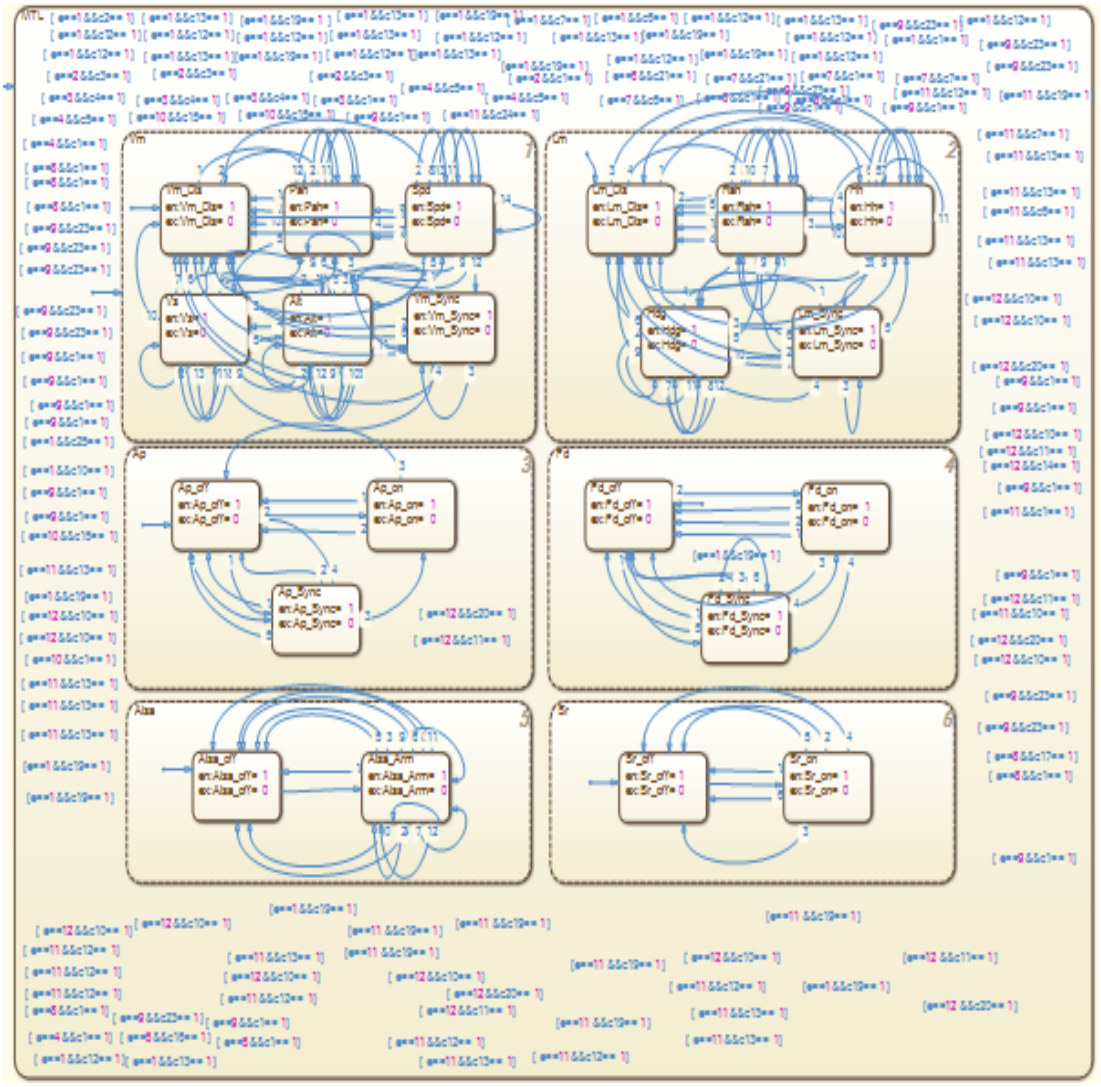


Figure 5: State flow chart of MTL.

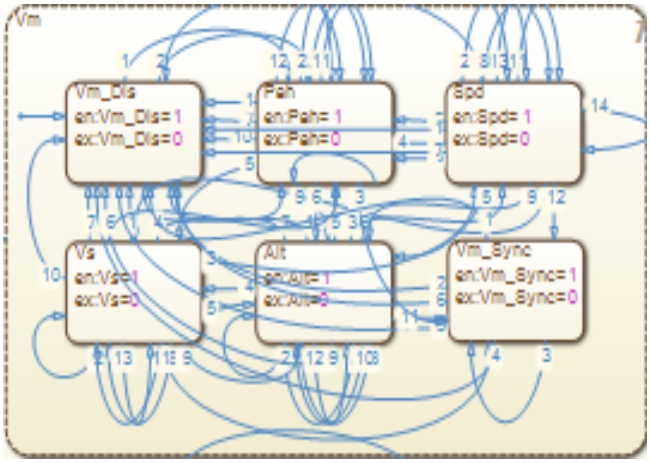


Figure 6: State flow chart of vertical mode.

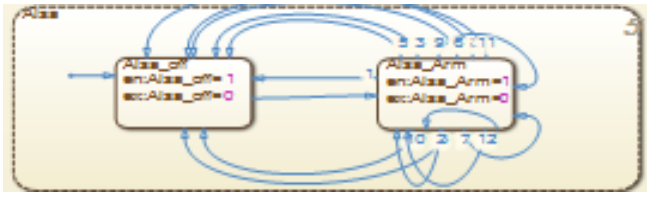


Figure 7: State flow chart of altitude select arm.

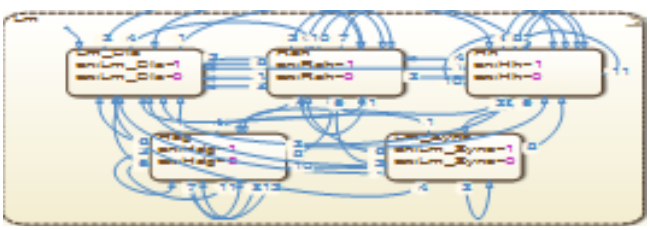


Figure 8: State flow chart of lateral mode.

from ARINC 429, and discrete sources which are acquired from external sensors in the aircraft. State transition logic is generated based on the pre-defined inputs, events, conditions, and allowed transitions.

Validation of the developed stateflow [version 8.1,] is performed at the model level as well as at the code level. At model level, the test scenarios to validate the model are translated from code level test cases. Test cases generated are imported to the matlab workspace and the outputs are analyzed with reference to the expected output [11]. Simulink design verifier [version 3.5,] is also used to generate the test scenarios and capture the test outputs to check the model functionality, performance, safety and stability. These outputs are compared to expected output [12] for the correct implementation of the MTL logic.

Validation at the code level is done by generating the autocode of the MTL stateflow. The in-house and approved code standards are used to generate a compact and safer autocode. Autocode generated from stateflow are imported to RTRT (Rational Test Real Time) tool suite, Version 7.5. Test cases generated for manual code are used for the autocode and analyzed for code functionality, code compactness and coverage. RTRT tool generates the coverage report. In case of safety critical software developed for highest criticality i.e., Level A [3], code is tested for 100% MCDC (modified condition decision coverage) coverage. Model level robustness is validated by importing the autocode into Reactis tool suite [13]. Reactis tool suite provides three types of analyses: Model Coverage Analysis, Reactis Tester and Reactis Validator. Test-cases for the autocode are generated automatically. The coverage report of Reactis Tool suite is compared with the model level and RTRT level test report. All three coverage reports are used in the validation of the MTL logic.

Implementation of MTL using Stateflow

Autopilot mode transition logic is implemented using MATLAB Stateflow. MTL inputs, conditions, and outputs are taken as a Boolean. Transitions are allowed only if the conditions are satisfied.

MTL logic is implemented using a hierarchical approach from top to bottom using a sequential implementation of the logic as:

» Top-model : provides the MTL architecture

» Vertical Mode : provides the logic for entry, exit of vertical modes for the autopilot

» Lateral Model: provides the logic for entry, exit of lateral modes for the autopilot

Figure 3 describes the software logic of the MTL which is to be translated into the stateflow model. Transition matrix guides the change of state on receiving an input from external interfaces. In order to successfully execute a state change on receipt of an input, certain conditions have to be met which is dictated by a condition matrix. Conditions check for values of certain inputs to be within specified ranges and if the condition is met, state is changed as per the transition matrix. State change also results in an output, as dictated by the output matrix. This output is used to command the flight control surfaces.

Top level model

The top-level model provides the MTL architecture. The control flow of the MTL logic is designed at this level. Figure 4 shows the top-level model of the autopilot MTL. Figure 4 consists of states, events and outputs. The state is a uniquely defined mode variable which can take certain discrete values. The state changes in response to external event and the respective conditions.

During the autopilot, when an event is pressed on the Autopilot Computer and Mode Selection Panel (ACMSP), the control (control matrix) flow checks the pre-defined conditions (condition matrix) for the MTL. If the condition is allowed then the transition (transition matrix) takes place to the allowed state, else the present state is retained and provides the respective output.

Figure 5 depicts different aircraft modes: Vertical Mode (VM), Altitude Select Arm (ALSA), Lateral Mode (LM), Autopilot (AP), Soft Ride (SR) and Flight Director (FD). This is the complete state diagram of MTL logic and it contains all the states, conditions and events as per the given inputs. Each of the state is explained separately in the following sections.

Vertical mode: The basic vertical mode is the (Pitch Altitude Hold) PAH. The transition to the higher level modes like the ALT, SPD, and VS Hold takes place when thecorresponding events conditions are satisfied. Figure 6 shows the state transition chart of vertical modes. The transition to the next state depends on the Event as well as Condition(s). If no event takes place the present state is maintained. The allowed modes are Vm_Dis, Spd, Vs, Alt, and Vm-Sync.

Altitude select arm: The Altitude Select arm (Alsa) is a compound vertical mode allowing the pilot to climb or descend to a pre-selected altitude and hold that altitude. The Alsa is armed by pressing the Alsa button on the ACMSP. Alsa gets engaged only when a specific condition is true, else remains disengaged. Figure 7 shows logic and allowed the state transition state for altitude select arm Alsa. It has only two modes: Alsa_arm, and Alsa_off.

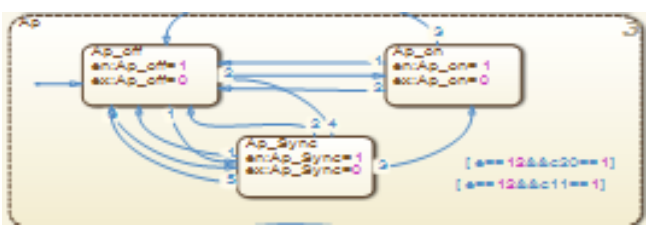


Figure 9: State flow chart of autopilot.

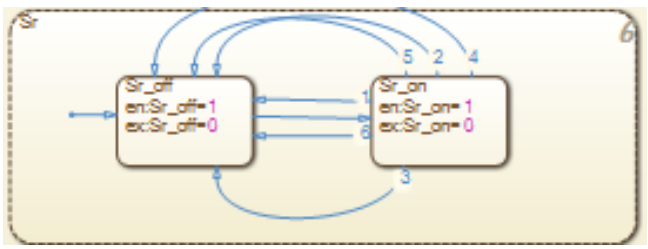


Figure 10: State flow chart of soft ride.

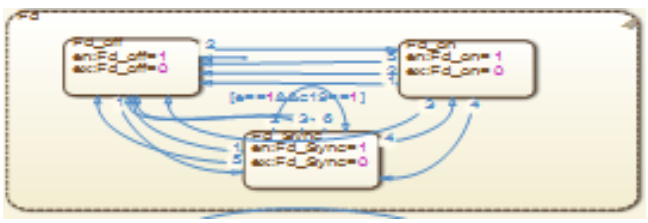


Figure 11: State chart of flight director.

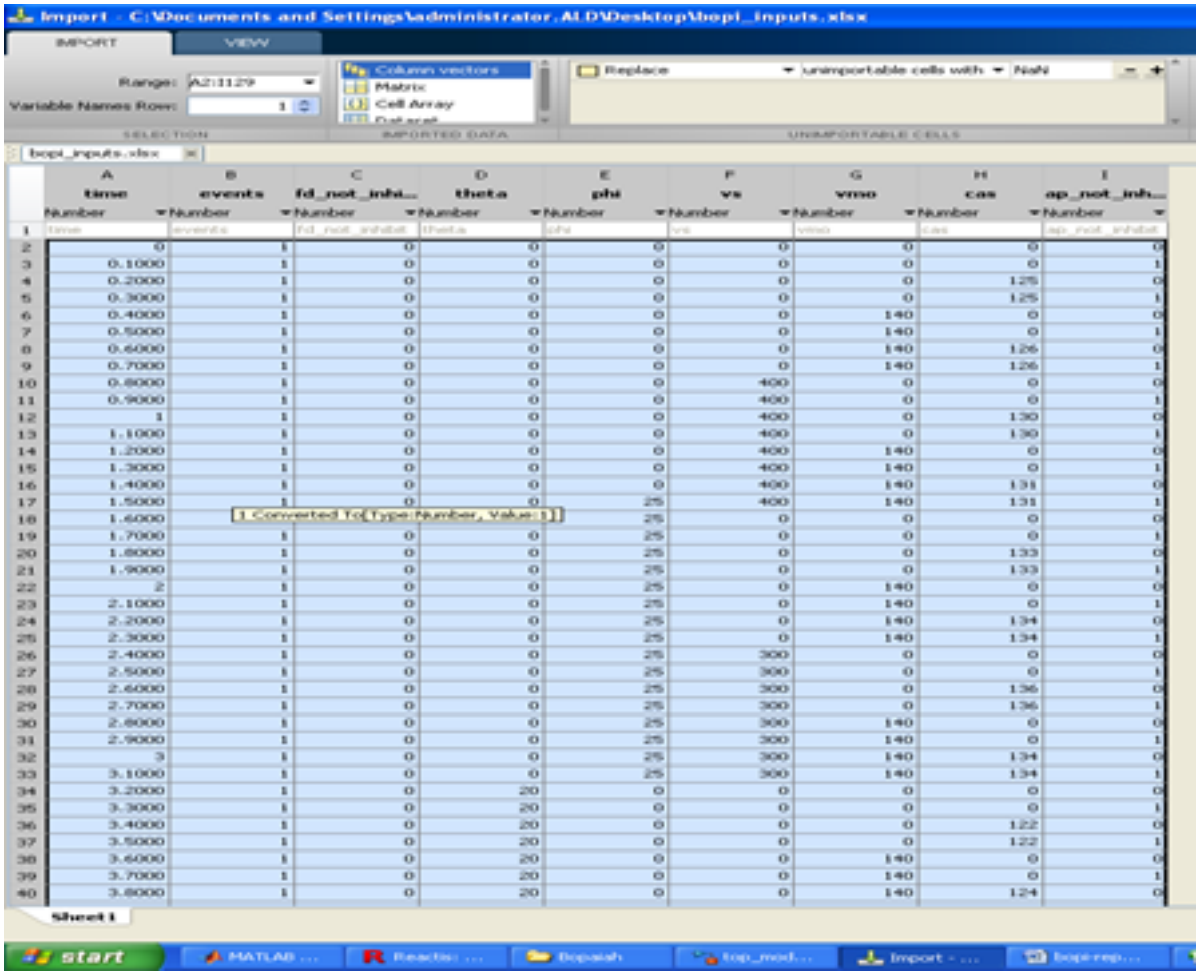


Figure 12: Imported inputs in Matlab environment.

Lateral mode: The basic lateral mode is the (Roll Attitude Hold) RAH. The transition to the higher-level modes like the HH Hold and HDG Select takes place when the corresponding event and conditions are satisfied. If no event takes place the present state is maintained. Figure 8 shows the logic and the allowed state transition chart for lateral mode. The allowed modes are: Lm_Dis, Hh, Hdq, and Lm_Sync.

Auto-pilot: The autopilot gets engaged only when the autopilot button on the ACMSP is pressed where the transition from off-state to on-state takes place, else the mode is in disengaged or off state. Figure 9 shows logic and allowed state transition chart for autopilot.

Soft ride: This mode is selected while encountering turbulence/ gusts in flight. On selecting this mode, any previously held higher mode is dropped and the basic modes (PAH, RAH) are engaged with the inner loop control law gains appropriately reduced to alleviate the effect of external disturbances. Figure 10 shows the logic and allowed state transition chart.

Flight director: Flight Director, FD, function computes the reference commands for the AP function based on the pilot selection of modes through the ACMSP and the aircraft motion parameters obtained from the AHRS and ADCU. The FD also computes the reference commands, which drive the steering bars on the EFIS for Flight Director Guidance. Figure 11 shows the logic and the allowed state transition chart.

MTL Model Design Validation

MTL model design validation was done at the model level, code level and cross validated using third party tool. Validation at model level is performed using the simulink/ stateflow environment. Validation of the autocode generated from the model was done using a third party tool.

Model validation using Simulink/ Stateflow environment

Matlab simulink/stateflow outputs: Simulink stateflow is validated for the functionality and robustness by providing the test cases. The test cases used for the manual code are imported from Excel to Matlab workspace. Figure 12 shows the imported inputs from Excel to Matlab workspace. In the matlab workspace inputs are given in particular timed interval for that particular time and the outputs are displayed in the same workspace. These outputs are used to compare with the experimental results. Figure 13 shows the corresponding outputs as per the imported inputs [20].

Figures 12 and 13 show the inputs and outputs for event 1 i.e., Bap, with 7 inputs and 21 outputs. Time column in the inputs shows the

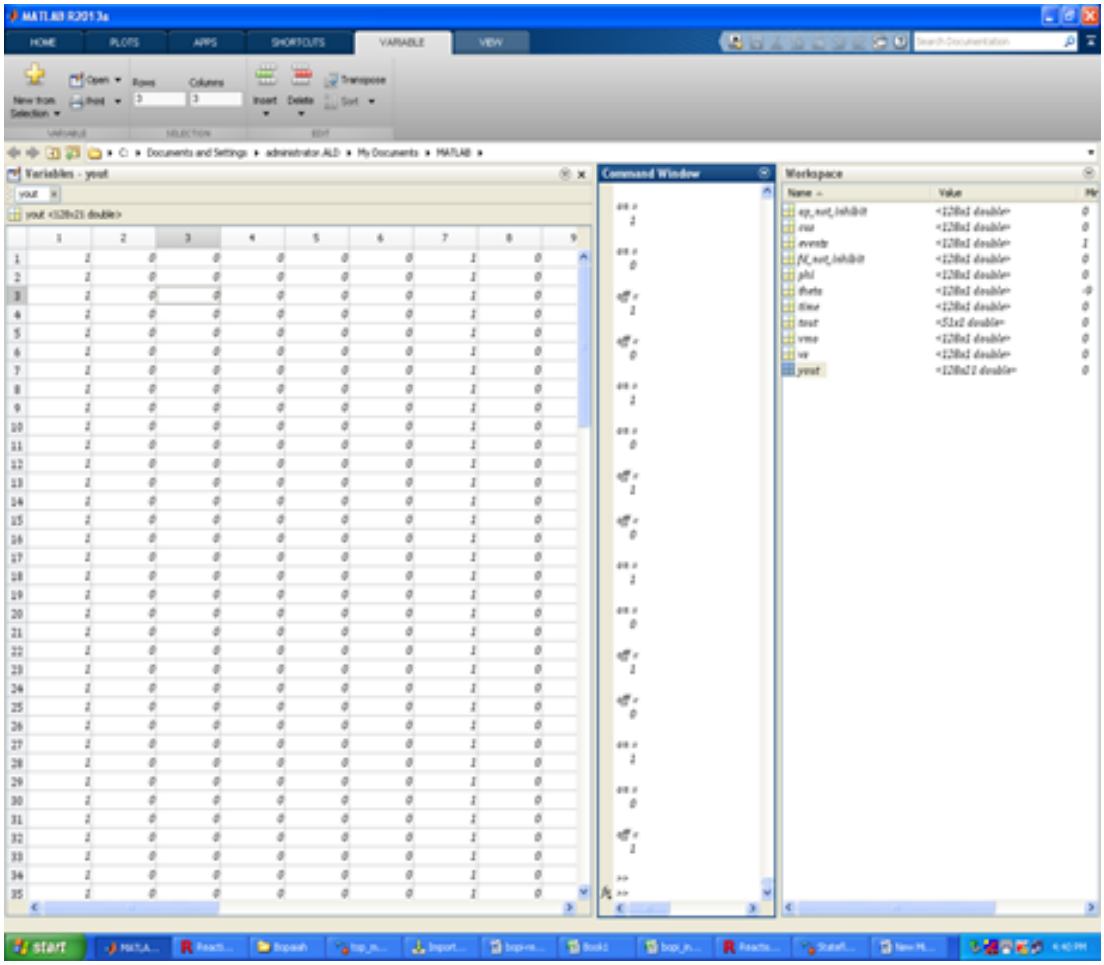


Figure13: Corresponding outputs in Matlab workspace.

specific time at which the corresponding row of inputs loaded into the model [21]. Other MTL modes are tested using similar approach.

Simulink design verifier (SlDV) [20]: Simulink Design Verifier™ uses formal methods to identify hidden design errors in models without extensive simulation runs. It detects blocks in the model that result in integer overflow, dead logic, array access violations, division by zero, and requirement violations. For each error it produces, there is a simulation test case for debugging. The Figure 14 is shows simulink design verifier report for mode transition logic. The logic is executing all the blocks of simulink/stateflow model. Totally 334 test cases have been used to cover the MTL logic.

Figure 15 shows the model advisor report. Model is built in Matlab Simulink/Stateflow Environment according to MAAB standard. Figure 16 shows the model advisor report of MTL. The report shows 10 fail standard and 13 warning because of model is not connected with hardware and internet to check the online resources with math work technical team.

Model checking or model advisor is an automated approach to verify that a model of a (usually concurrent, reactive) finite state system satisfies a formal specification of requirements to the system. In this approach how the system are behaving and generating the test cases automatically to analyze the behavior of the model. Tools that automatically perform model checking are called model checkers.

Rational test real time (RTRT) analysis

Rational Test Real Time is a cross-platform solution for component testing and runtime analysis. It is designed for developers creating complex systems for embedded, real-time and other cross-platform distributed applications. This software helps you debug and correct errors before they go into production code. RTRT resolve software problems during the development phase - allows testing the components. It can analyze the performance and reliability of the applications as run on the host development system. Modeling the system in modeling tool and generate the corresponding C/C++ code. This autocode was imported into the RTRT environment and test cases are written as per the requirements.

Figure 16 shows the 100% coverage of functions, functions and exits, statement blocks, decisions, basic conditions, modified conditions, multiple conditions in percentage. Figure 17 shows the report summary of RTRT unit testing with zero failure. It's totally having 212 test cases to cover the auto code.

Model validation-rectis environment

Reactis offers model-based testing, debugging, and validation for Simulink / Stateflow models. Reactis currently consists of three main

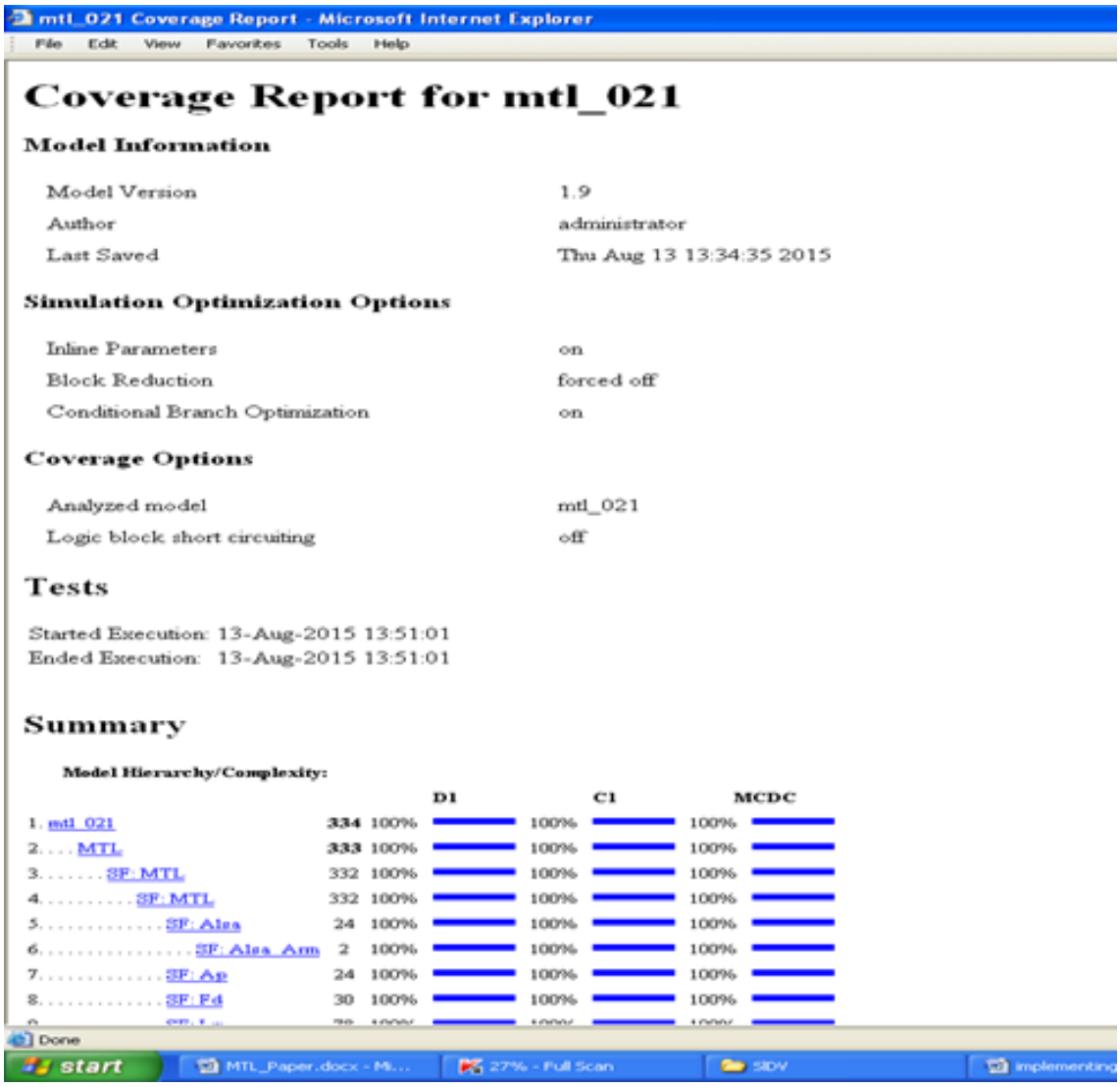


Figure14: Simulink design verifier report.

components: Reactis Tester, Reactis Simulator, and Reactis Validator [21,22]. Reactis Tester automatically generates test suites from Simulink / Stateflow models of embedded control software. The test suites provide comprehensive yet concise coverage of different test-quality metrics. Each test in a test suite consists of a sequence of input vectors as well as the responses to those inputs generated by the model. These tests may be used for a variety of purposes, including [23-25].

- **Implementation conformance.**

The tests may be applied to implementations derived from models to ensure conformance with model behavior.

- **Model testing and debugging.**

The tests may be run on the models themselves to analyze model behavior and to detect runtime errors.

- **Regression testing.**

The tests may be run on a new version of a model to compare its behavior to an older version.

- **Reverse engineering of models from source.**

Tests may be generated from models derived from legacy code in order to check conformance between model and code. Reactis enables to maximize the effectiveness of testing while reducing time and effort.

Reactis coverage: Figure 18 shows the MTL coverage report in Reactis environment. Reactis generated test cases automatically and executed the model and code satisfying the decision, conditions and MC/DC 100% [21]

Reactis validator: Figure 19 is report for reactis validator analysis. Reactis validator analysis is used to validate the simulink model in the reactis environment and it shows the validator coverage report is 99% is true [21].

Reactis tester: Figure 20 shows the report of reactis tester. Reactis tester is testing the model as per the input assigned in the property and generated the coverage report. The report shows the 99% of true in test cases of the MTL logic. Those test cases are covering decision 100%, Condition 100% and MC/DC 98%. Those test cases coverage report

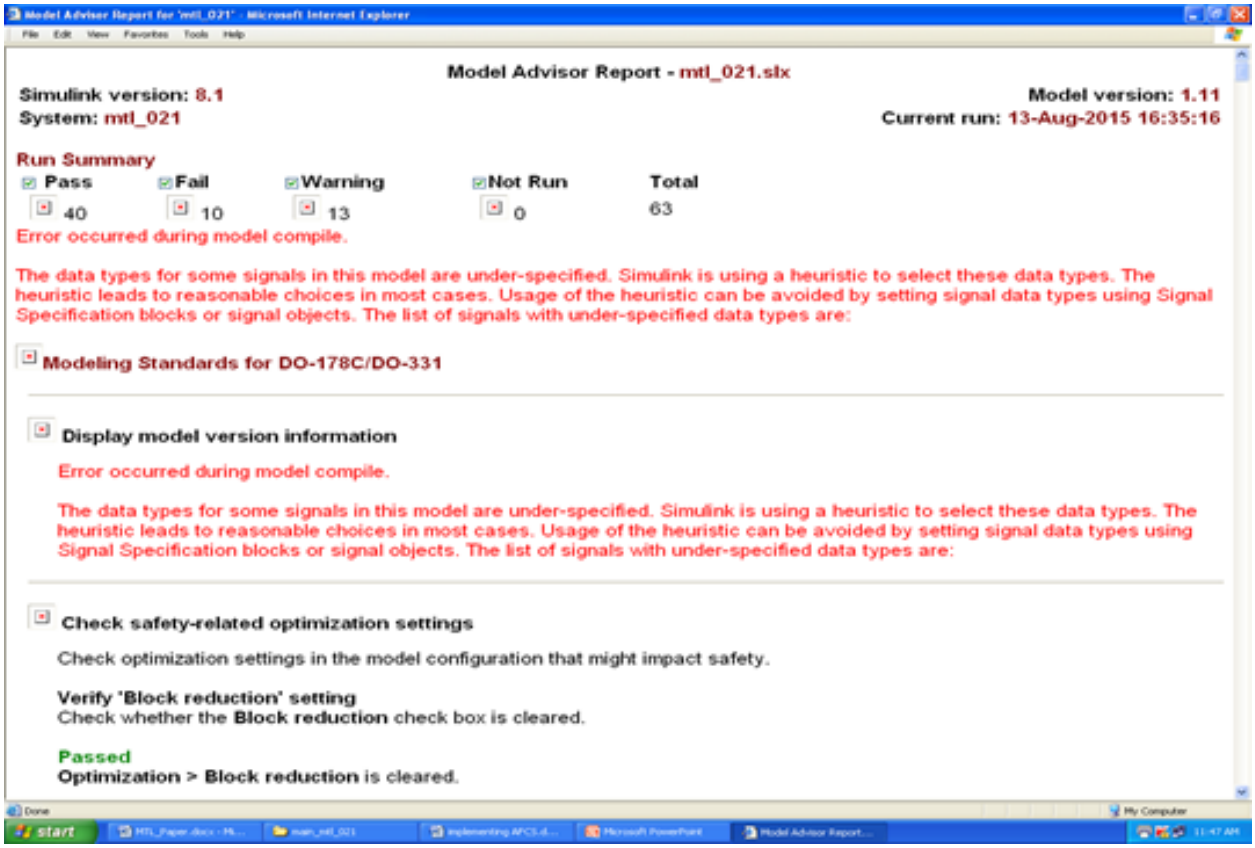


Figure15: Model advisor report.

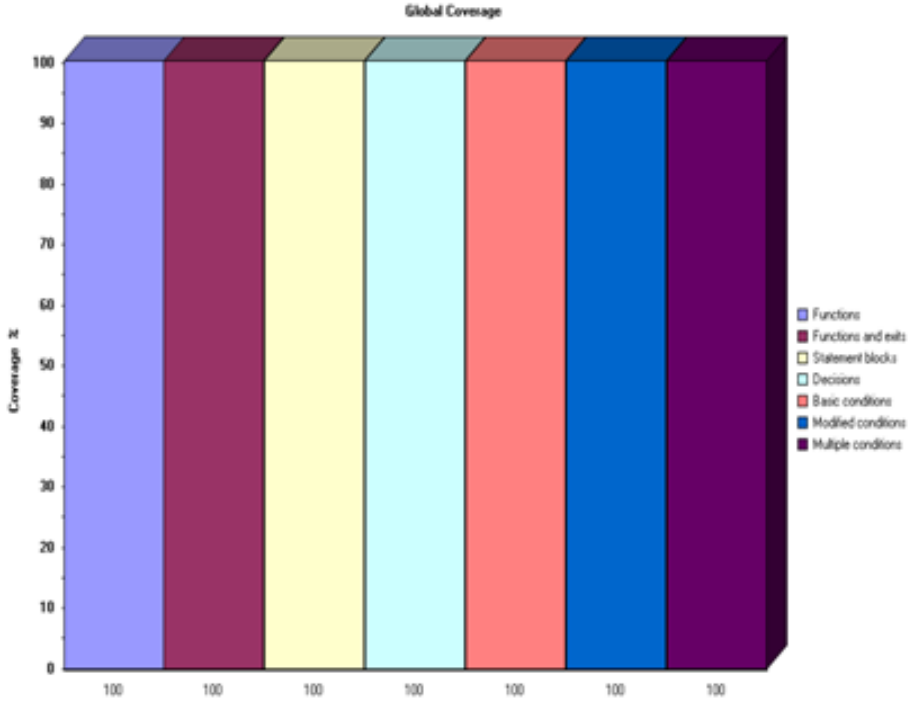
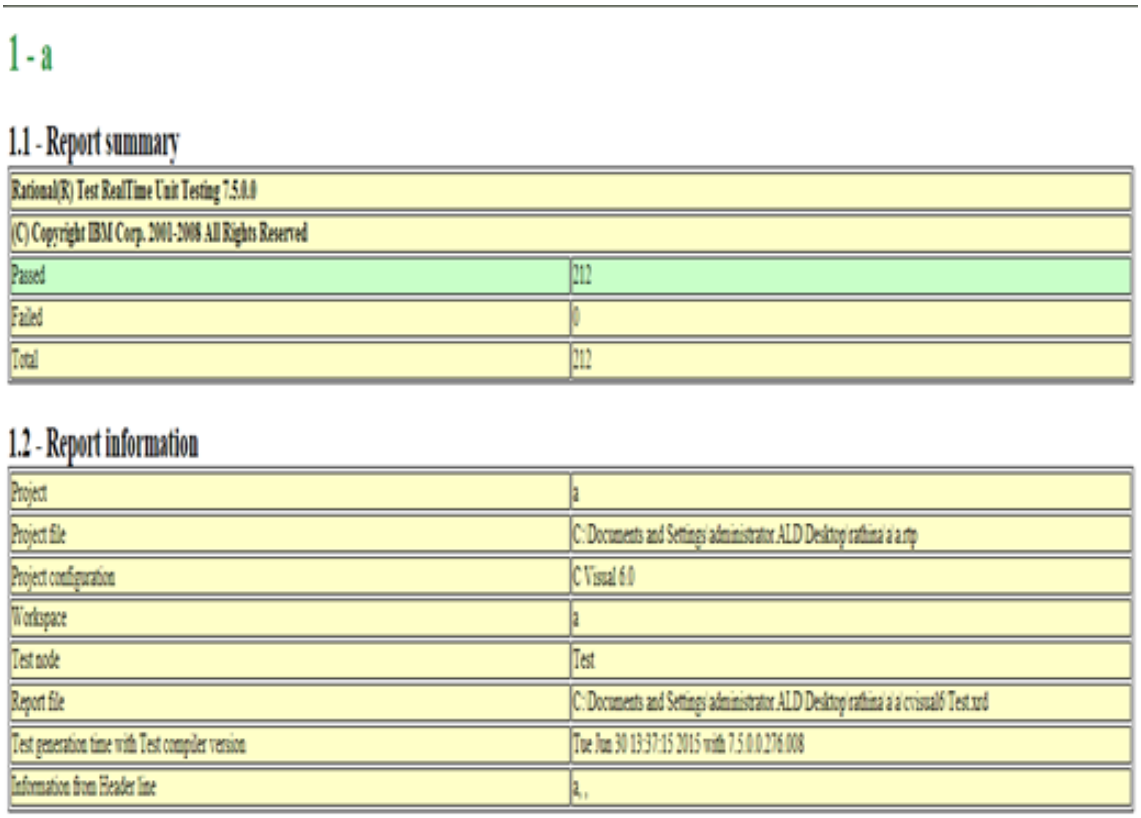


Figure 16: Global coverage of MTL code.



1 - a

1.1 - Report summary

Rational(R) Test RealTime Unit Testing 7.5.0.0	
(C) Copyright IBM Corp. 2001-2008 All Rights Reserved	
Passed	212
Failed	0
Total	212

1.2 - Report information

Project	a
Project file	C:\Documents and Settings\administrator.ALD\Desktop\rathina\a\a.rtp
Project configuration	C Visual 6.0
Workspace	a
Test mode	Test
Report file	C:\Documents and Settings\administrator.ALD\Desktop\rathina\a\a cvisual6 Test.rsrd
Test generation time with Test compiler version	Tue Jun 30 13:37:15 2015 with 7.5.0.0.276.008
Information from Header line	a, ,

Figure17: Report summary of MTL code.

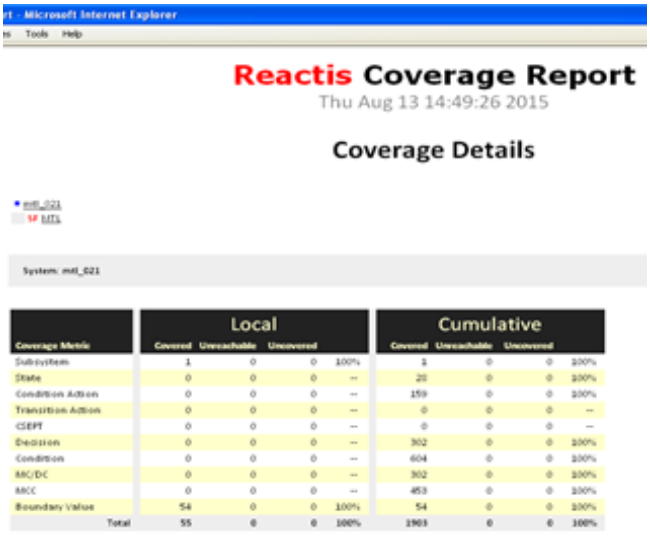


Figure 18: Rectis coverage report.

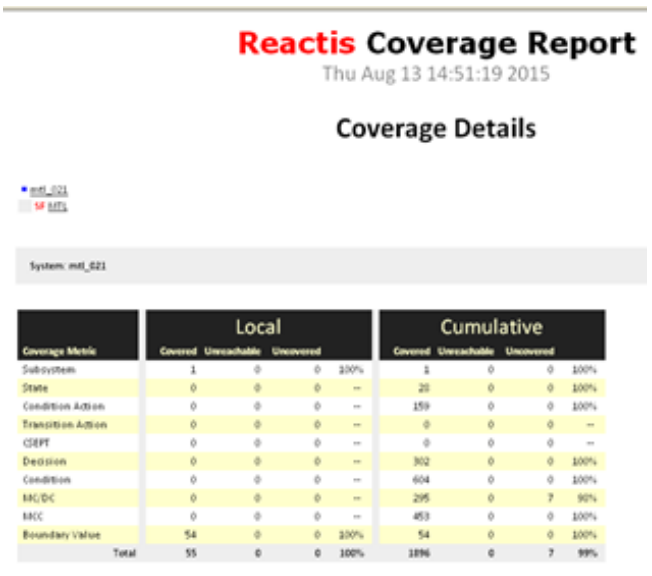


Figure 19: Rectis validator report.

is available in rectis .rst file. Figure 20 shows only the main coverage report of Reactis tester [21].

Result Analysis

Table 1 consolidates the MTL validation result carried out at model and code level. The MTL logic is implemented using stateflow and autocode is generated for the stateflow. The correct and complete implementation is validated using various complementing techniques. At the model level, SlDV and Reactis are used to validate the model as per the requirements. At the code level, RTRT is used to validate the code. The report generated can be used as artifact for the adherence to RTCA DO-178C certification of complex logic.

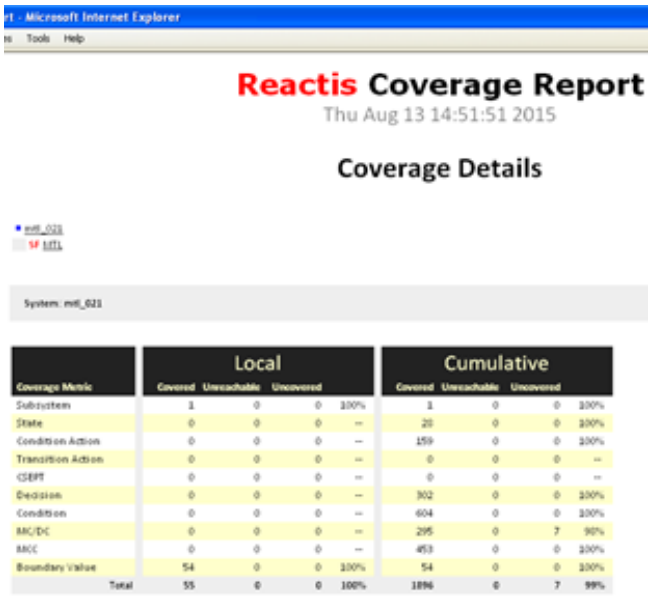


Figure 20: Rectis tester report.

Environment	Decision	Condition	MC/DC
Matlab	100%	100%	100%
(SIDV)	100%	100%	100%
RTRT	100%	100%	100%
Reactis Coverage	100%	100%	100%
Reactis Validator	100%	100%	98%
Reactis Tester	100%	100%	98%

Table 1: Final result analysis comparisons.

Conclusion

One of the effective ways to implement the complex logic is by means of formal methods. Semi-formal method based tools help in visualizing, realizing and validating the complex logic. This approach is more effective than the manual approach where the individual needs to verify based on ones experience. The implementation and validation of the complex MTL logic using semi-formal method approach demonstrates this. As the implementation was carried out at the model level, the semantic design flaws are uncovered earlier in the engineering process. The understanding of such complex logics is better understood using stateflow in comparison to tabular information. The proposed approach provides methods to validate the MTL for its correct and complete functionality. The traceability of the stateflow can be generated to the requirements and code if required.

Work ahead is to perform the safety analysis of the MTL at the model level. This will not only provide the functional robustness but also safety properties of the logic at the model level. The Safety analysis integration will provide the failure behavior information of the system. The modified approach will integrate safety analysis to a integrated model-based formal analysis of complex software designs.

The Simulink stateflow is an effective way to model the complex logic of aircraft mode-transition-logic. In this paper the transitions from one state to another using formal method has been described. All the possible mode transitions in the presence of external event and condition(s) are presented in the stateflow chart, which is easy to understand, analyze, debug and generate source code.

The analysis of the MTL for its functionality & safety is performed at the model level. The outcome of the approach is encouraging approach to adopt semi formal methods in other safety critical application. The proposed approach not only reduces the design effort but also provides higher assurance in the design and functionality of the complex system such as the autopilot. The model-level test cases generated using formal techniques can be translated to the code test cases to ensure traceability of code and model.

Acknowledgment

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

References

1. Boorman DJ, Mumaw RJ (2004) A new autoflight/FMS interface guiding design principles.
2. Randhawa P, Mishra A, Jeppu Y, Nayak CG, Murthy N (2012) Mode transition logic of a vertical autopilot for commercial aircrafts IX control instrumentation system conference (CISCON-2012) :16-17.
3. Nair AS, Jeppu Y, Nayak CG (2013) Logic for mode transition of autopilots in lateral direction for commercial aircrafts bonfring. Int J Man Machine Interface.
4. (1992) RTCA DO-178B Software considerations in airborne systems and equipment certification RTCA Inc Washington DC.
5. Meenakshi B, Barman KD, Babu G, sehgal K (2007) Formal safety analysis of mode transition in aircraft flight control system .
6. Joshi A, Miller SP, Heimdahl MPE (2003) Mode confusion analysis of a flight guidance system using formal Methods.
7. Littegen G, Carreno V (1999) Analyzing mode confusion via model checking, NASA langley research center, Hampton, Virginia 23681-2199,USA.
8. Degani A, Heyamann M (2000) Pilot-autopilot interaction: A formal perspective. 8th International Conference on Human – Computer Interaction in Aeronautics :1-11.
9. El-Gendy H, El-Kadhi N (2005) Formal methods: Important experience and comparative analysis. J Computational Methods In Sciences And Engineering 5: 235-247.
10. Lutz RR, Ampo Y (1996) Using formal methods for requirements analysis of critical spacecraft software IEEE 1-6.
11. Ait Ameur Y, Boniol F, Wields V (2007) Towards a wider use of formal methods for aerospace system design and verification. Int J Tools transfer 12:1-7.
12. Kuhn DR, Craigen D, Saaltink M (2003) Practical application of formal methods in modeling and simulation. National Institute of Standards and Technology.
13. Kececi N, Halang WA, Abran A (2002) A semi-formal method to verify correctness of functional requirements specifications of complex embedded system.
14. Gavrilets V, Martinos I, Mettler B, Feron E (2002) Control logic for automated aerobatic flight of a miniature helicopter. American Institute of Aeronautics and Astronautics Inc AIAA Guidance Navigation and control conference and exhibit AIAA.
15. Yadav A, Gaur P (2014) AI-based adaptive control and design of autopilot system for nonlinear UAV. Indian Academy of Sciences 39: 765-783.
16. Aldrich W (2002) Using model coverage analysis to improve the controls development process. AIAA Modeling and Simulation Technologies Conference and Exhibit.
17. Balachandran S, Atkins EM (2015) Flight safety assessment and management during take-off. AIAA.
18. Busser RD, Blacburn MR, Nauman AM Automated model analysis and test generation for flight guidance mode logic. IEEE Xplore Digital Library.
19. Ming Z, Yung L, Yi Q, Xiongwen H, Xiaochuan Z et al. (2012) Model Based Design UAV Autopilot software. Proceeding of the 2012 2nd international conference on Computer and Information Application (ICCIA 2012) published by Atlantis press Paris France.
20. (2015) Mathworks Inc. Simulink/Stateflow.

21. (2015) Mathworks Inc. Model Advisor.

22. Reactis System Inc. Reactis validation tester and analyzer.

23. Cofer D, Whalen M, Miller S (2008) Software model checking for avionics systems.

24. (2015) Mathworks Inc. Simulink Design Verifier.

25. Book: Rational IBM Rational Test Real Time RTRT-User-Guide for version 7.0 G|11-6755-00.