Formal Architecture Based Design Analysis for Certifying SWS RTOS

Yalamati Ramoji Rao, Manju Nanda and J. Jayanthi

Abstract In recent times Formal Techniques have been strongly recommended in the engineering life-cycle of safety -critical systems. With this, Architecture Analysis & Design Language (AADL) is a widely spectrum accepted architecture modeling language that can be wrap with Formal Modeling techniques, that proficiently helps in the design of a safety-critical system and circumscribes various analytical features for modeling the hardware and software architecture/s, against the required as per the guidelines set aside in RTCA DO-178C (333- Formal Based Modeling). This paper discusses the use of architecture modeling language along with formal based techniques for the analysis of RTOS architecture which is important in the correct implement of the given requirements. The architecture of the RTOS is expressed and analyzed using AADL. A suitable case study such as Stall Warning System/Aircraft Interface Computer (SWS/AIC), RTOS scheduler is modeled and analyzed. The analysis of results are mapped to the workflow prescribed in RTCA DO-178C for generating the certificate artifact and establishing the effectiveness of architecture based design analysis in the software engineering process.

Keywords Safety-Critical system · Multi-function · RTOS · Formal method · Architecture analysis & design language (AADL) · SWS/AIC · Certification artifacts

1 Introduction

Avionics architecture means set of various electronics operated in aircrafts. Since 70's avionics architecture is composed of several digital and communication

Y.R. Rao
Institute of Science and Technology, JNTUK-Kakinda University,
Kakinada, Andhrapradesh, India
e-mail: ramojirao999@gmail.com

M. Nanda(✉) · J. Jayanthi
CSIR- National Aerospace Laboratories, Bangalore, India
e-mail: {manjun,jayanthi}@nal.res.in

S. Berretti et al. (eds.), *Intelligent Systems Technologies and Applications*,
Advances in Intelligent Systems and Computing 385,
DOI: 10.1007/978-3-319-23258-4_38

systems which support more and more avionics applications such as flight controls, and flight management system's to name a few. Over the years as the demand is increasing, the avionics system architecture is becoming complex and central component of an aircraft. In case of civil aerospace 30 to 40 % of development cost is due to avionics system whereas in case of military it goes to 40 to 50% [1]. The higher cost of avionics system is because it involves more R&D work for each and every system that has been installed in aircraft. This cost can be reduced if we analyze the system earlier in the phase i.e. before the design level which is the Architecture level. Architectural analysis not only helps in detecting the design flaws earlier but it also helps in understanding the system requirements.

RTOS plays a critical role in realizing the real time functionalities of the Safety-Critical Systems. The scheduling feature of the RTOS provides the timeliness of the embedded systems. In this paper, we propose an approach to analyze the architecture of the RTOS. Analysis of RTOS helps in identifying potential risks of the scheduler against the requirements [2]. Developing a reliable RTOS architecture for a complex system is a critically important step for ensuring that the system satisfies its principal objectives. Examples of safety critical RTOS includes QNX, VxWorks, DEOS, and LynxOS [3].

The paper provides an approach to analyze the RTOS scheduler. This is done by means of a case study of the SWS/AIC RTOS scheduler. The RTOS scheduler is modeled using the AADL using the OSATE plug-in in Eclipse IDE. The scheduler timeliness, processor utilization and response time features are analyzed as per the project requirements. The analysis results are mapped to the RTCADO-178C/ DO-333 workflow to establish certification of RTOS using formal methods.

2 Literature Survey

2.1 Complex Safety-Critical Systems

Advanced avionics systems are computer centric systems which are used to achieve the desired functionality. Failure in these systems endangers the human life and large scale of economics [5][6]. Functionality within the avionics software continues to expand. Additional software capabilities bring many more lines of code, and generate opportunity for error. As the complexity requirements and criticality of avionics software grow, innovative tools are used to test, verify and secure the architecture of such systems." Safety-Critical Systems has to be bug free, and it has to work according to specifications", that's where the FAA's RTCA DO-178B(C) standards comes in. Static analysis software tool analyze source code to derive properties that can helps in detecting the errors that might not be apparent to the programmer, while dynamic analysis tool helps in showcasing which code is executed by the test suite. "All have their place in safety critical development", for a complex safety critical avionics software, the requirements for testing and validation are higher than most of other software [7][8][9][10][11].

2.2 RTOS

Real-Time Operating Systems (RTOS) are required to provide a predictable platform for the execution of multiple software tasks on single microprocessors.Every RTOS has a kernel, which serves as its core. The core is surrounded by shell layers, which provide protection and access authorizations. The RTOS manufacturer provides a set of APIs, which are used to access this kernel to perform tasks. The kernel contains a scheduler to execute these threads in a sequential manner. RTOSs are of three types; Hard real-time, Firm real-time and Soft real-time. In a hard real-time system the tasks and interrupt requests must be completed within their required deadlines. Failure to meet a single deadline may lead to a critical catastrophic system failure. Firm-real time system tolerates the missing of a few deadlines; however, missing more than a few may lead to complete disaster. A soft real-time system is one in which deadlines are mostly met, but this constraint is not very stringent. RTOS must have sufficient number of priority levels and must avoid priority inversion. RTOS Scheduler is the most critical component as it ensures the timeliness of the tasks as per the requirements [3].

2.3 Formal Methods

Nowadays the advantages of formal methods for the development and certification of safety-critical software are widely recognised by both Software Engineering community and standard organisations [18]. Some safety critical domains specific standards explicitly either highly recommend or mandatory require the use of formal methods. The specific benefit recognised to formal methods is that they allow "complex behaviour" to be analysed (by means of proofs or state exploration), reviewed, and analysed in their totality, rather than merely sampled as by testing or simulation. Thus, the major benefit derives from a double application of formal methods that is by formal requirements specification coupled with formal verification. From the certification point of view, formal methods are recognised to increase the degree of confidence in achieving software of high integrity levels [23]. Formal methods that are able to address real-time issues offer a unique opportunity for the specification of such time critical operating systems[20].The advantage of using formal methods in avionics software is because of

- *Reduced Cost* : Early detection/Elimination of defects
- *Increase Confidence*: Complete examination of models and requirements
- *Satisfy certification Objectives*: RTCA DO-178C [19]

3 Proposed Methodology

3.1 RTOS Kernel Overview

In this paper we propose a methodology to validate the RTOS architecture of a proven safety-critical system, Stall Warning System / Aircraft Interface Computer

(SWS/ AIC)kernel architecture. The proven system is selected in order to prove the effectiveness of architecture analysis and its certification impact with respect to the civil aerospace software standard RTCA DO-178C [16].

SWS/AIC system kernel is known asAPM2000 kernel. The scheduler schedules and dispatches the application tasks as per their grouping into fast, slow, and background tasks[21]. The complete processing of application is carried out through foreground and background tasks. The foreground task consists of fast, and slow tasks, fast task is executed every 25 msec. The slow task is executed every 100 msec. The fast task has higher priority than the slow task and background task. Background task has the least priority. At the startup sequence, the software manages all the interrupts to schedule and execute the application. The scheduling of SWS/AIC kernel is shown in Figure 1.

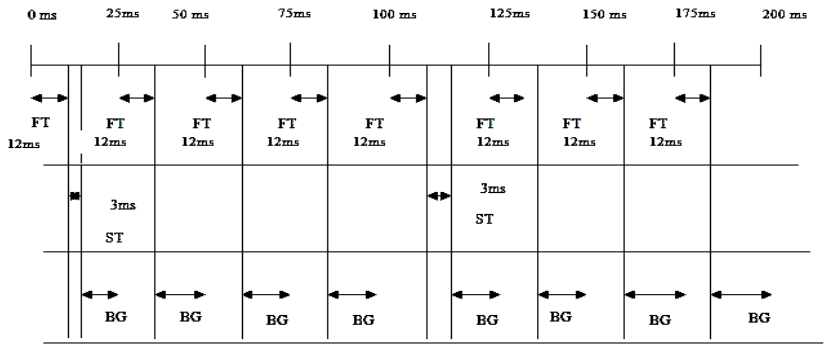


Fig. 1 Scheduling in the Stall Warning System

Note *- FT: Fast Task; ST: Slow Task; BG: Background Task*

In order to model the SWS/AIC kernel architecture as per the requirements we use AADL based architecture analysis approach. To design the kernel architecture detailed understanding of the FT, ST& BG was performed.

3.2 Fast Task Requirements and AADL Implementation

Fast task is given control by the kernel when the 25ms RTC interrupt occurs. This task has a higher priority than the slow and background task. The maximum task duration for the foreground task is 25msec. Only 60% of the foreground task will be utilized which is 60% of 25msec = 15msec. The design for the fast task duration is 80% of the 15msec, which is 12msec.In the fast task, excessive context switch and associated over-head is reduced by limiting the nesting of calls. The fast task utilization is such that the processor throughput is not affected. The activities in the Fast task include: Acquire inputs, CCDL Tx/Rx, ARINC output processing, Compute FT, Average Time, Discrete data validation, PTT on ground/air, WDT Kernel call, CBIT and Shaker stall Validation. These tasks are shown in Figure 2. The AADL model of the Fast Task is shown in Figure 3.

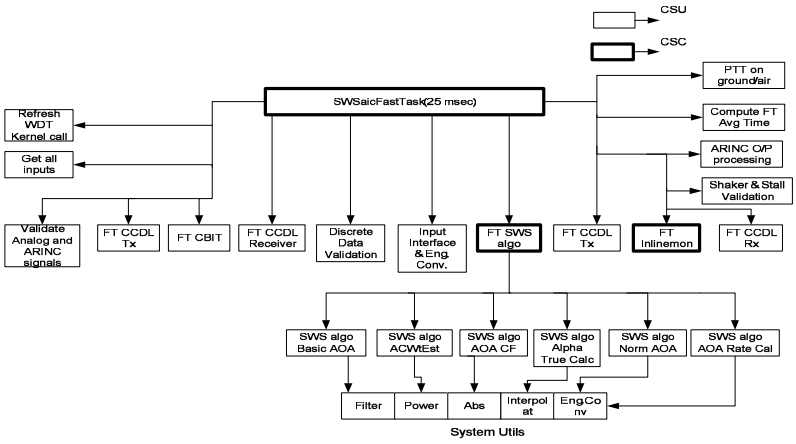


Fig. 2 Fast task Process Flow

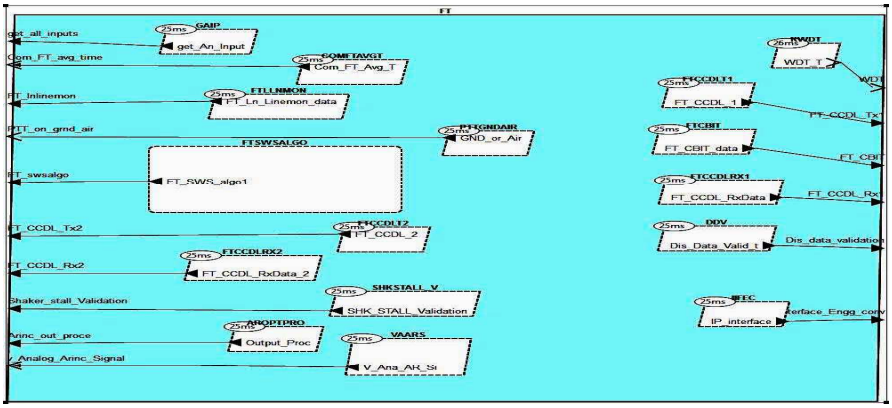


Fig. 3 AADL representation of fast task

3.3 Slow Task Requirements and AADL Implementation

Slow task in the foreground is given control by the kernel when the 100ms RTC interrupt occurs. This task has a higher priority than background task but lower than the fast task. The design for slow task duration is about 20% of 15msec = 3msec. The Fast Task interrupt can preempt this 100ms slow task. The slow task utilization is such that the process or throughput is not affected. The tasks in the slow task are: Output validation, output processing, Compute average time, & aic algo. These tasks are shown in Figure 4 and the AADL representation of slow task is shown in Figure 5.

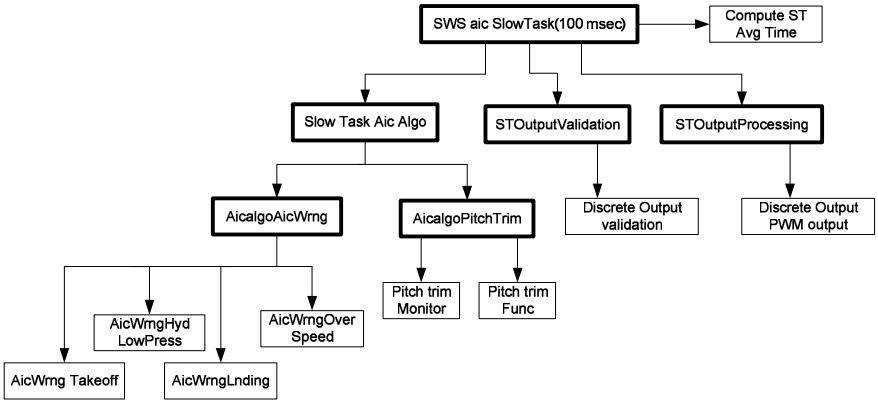


Fig. 4 Slow task Details

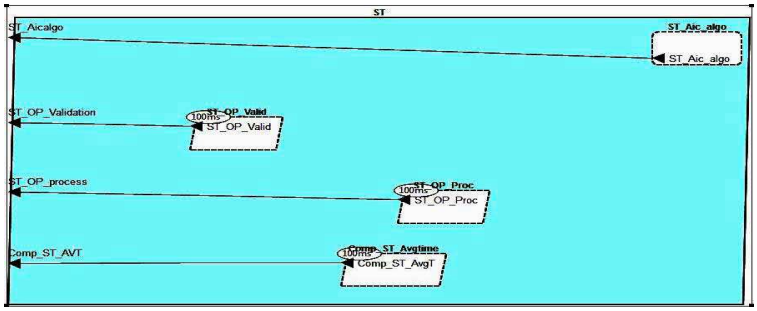


Fig. 5 AADL representation of slow task

3.4 Background Task Requirements and AADL Implementation

Background task is given control by the kernel after Power On and during the initialization of the system. It remains in this task until the 25ms fast task or 100ms slow task interrupt occurs. The fast task has the higher priority than the slow task. After the execution of fast task the kernel passes the control to slow task. After the completion of slow task, if no fast task/ slow task interrupt occurs only then the kernel passes the control to the background task. The tasks in the background task are: PBIT startup, CBIT, and Failed mode. The tasks are shown in Figure 6 and the AADL representation of background task is shown in Figure 7.

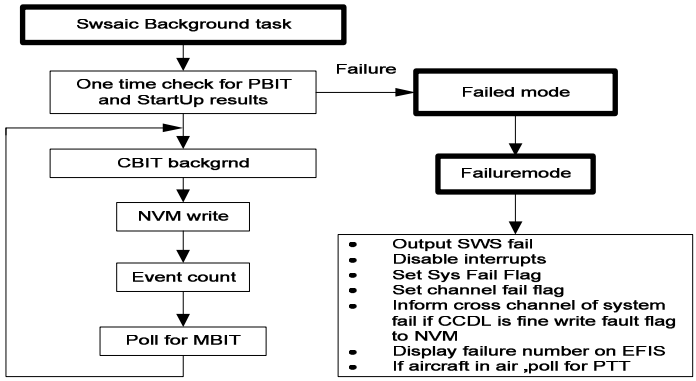


Fig. 6 Background task details

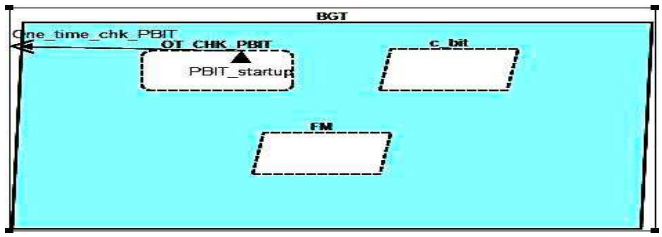


Fig. 7 AADL representation of Background task

4 Kernel Scheduler Simulation and Analysis

The representation of the architecture using AADL becomes easier to understand and implement as compared to the manual approach which gets finalized after multiple reviews.

The analysis of the AADL architecture provides model, application and execution statistics. The modeling and analysis using AADL is carried out using the OSATE Tool, Version 2.2[4].

The entire SWS/AIC kernel architecture is justified with evaluation of metrics such as: The *model statistics, application statistics, and execution platform. Model statistics involves* various aspects of model such as the component types, their declaration instances, and data flow types. The *application statistics* provides brief description about the software architecture involving thread instances, process instances, semantic connections between them, and end-to-end flow instances. The *execution platform* statistics provides clarification about the hardware instances involved in supporting the execution of the above justified instances such as processors and memory units that are bound during the execution of the software instances. The metrics generated by the tool is shown in Figure 8.

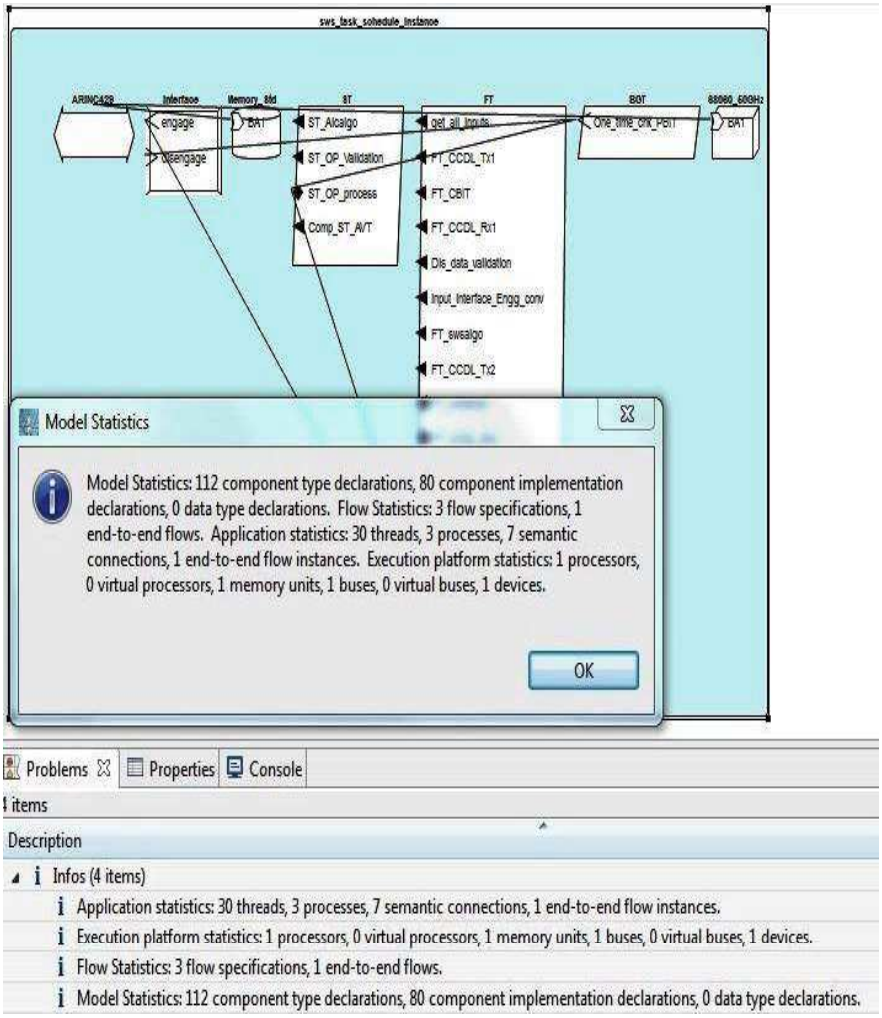


Fig. 8 Model Statistics of SWS/AIC Kernel Architecture

4.1 Kernel Scheduling Analysis

Analysis is the art of planning your activities so that you can achieve your goals and priorities as per the project requirements. Analysis helps in:

- Understand what you can realistically achieve with your time.
- Make sure you have enough time for essential tasks.

The kernel scheduling analysis using AADL Inspector provides cheddar schedulable table, Theoretical schedulable test and simulation schedulable test. Cheddar schedulable table provides the scheduling analysis of each and every task. The cheddar schedulable table for the SWS/AIC scheduler is in Figure 9.

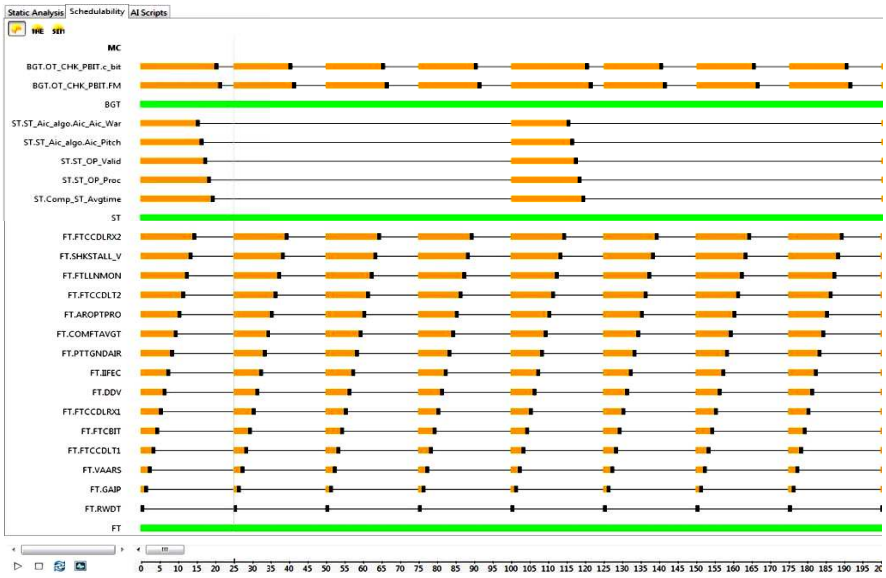


Fig. 9 Scheduling analysis of SWS/AIC Kernel

In Figure 9: Orange color represents thread is in ready state, Black color represents thread is in running state, Solid black line represents thread is in suspended state, and Green color represent process is in running.

Theoretical scheduling provides report on processor utilization factor with period & worst case task response time. The theoretical scheduler for the SWS/AIC is shown in Figure 10

test	entity	result
processor utilization factor	root.mc	Can not apply the feasibility test on processor utilization factor on this scheduler.
base period	all	100.00000
processor utilization factor with deadline	all	0.73000
processor utilization factor with period	all	0.73000
worst case task response time	root.mc	All task deadlines will be met : the task set is schedulable.
response time	root.mc.bgt.ot_chk_pbit.fm	22.00000
response time	root.mc.bgt.ot_chk_pbit.c_bit	21.00000
response time	root.mc.st.comp_st_avgtime	20.00000
response time	root.mc.st.st_op_proc	19.00000
response time	root.mc.st.st_op_valid	18.00000

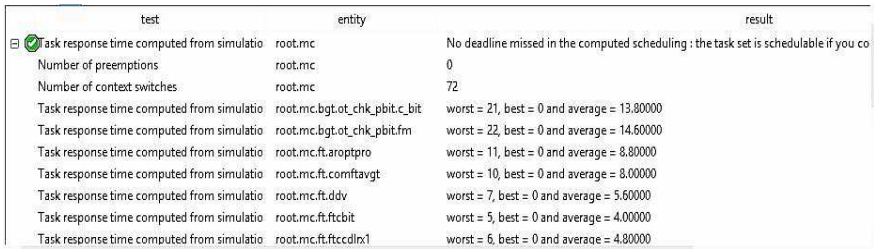
Fig. 10 Theoretical Scheduling of SWS/AIC Kernel

$$\textit{Processor Utilization}: U = \sum_{i=1}^{n} \frac{c_i}{p_i} \leq n.\left(2^{\frac{1}{n}} - 1\right)$$

n = No. of Tasks; C_i= Execution Time i^{th} task; P_i = Period of i^{th} task.

Fast task has C=15 msec, P = 25 msec; Slow task has C=5 msec, P = 100 msec; Background task C = 2msec, P = 25 msec; So U= 0.73 &B = $n.(2^{1/n}-1)$ = 0.77 since U $\leq$B deadline will be met[24].

Simulation scheduling provides task response time computed from simulation, number of preemption and number of context switches. The simulation scheduler for the SWS/AIC is shown in Figure 11.



test	entity	result
Task response time computed from simulatio	root.mc	No deadline missed in the computed scheduling : the task set is schedulable if you co
Number of preemptions	root.mc	0
Number of context switches	root.mc	72
Task response time computed from simulatio	root.mc.bgt.ot_chk_pbit.c_bit	worst = 21, best = 0 and average = 13.80000
Task response time computed from simulatio	root.mc.bgt.ot_chk_pbit.fm	worst = 22, best = 0 and average = 14.60000
Task response time computed from simulatio	root.mc.ft.aroptpro	worst = 11, best = 0 and average = 8.80000
Task response time computed from simulatio	root.mc.ft.comftavgt	worst = 10, best = 0 and average = 8.00000
Task response time computed from simulatio	root.mc.ft.ddv	worst = 7, best = 0 and average = 5.60000
Task response time computed from simulatio	root.mc.ft.ftcbit	worst = 5, best = 0 and average = 4.00000
Task response time computed from simulatio	root.mc.ft.ftccdlrx1	worst = 6, best = 0 and average = 4.80000

Fig. 11 Simulation Scheduling of SWS/AIC Kernel

Worst Case Response Time: $W_i^n = C_i + \sum_{j \in hp(i)} \left(\frac{C_i}{P_j}\right) . C_j$

C_i= Execution Time i^{th} task, P_i = Period of i^{th}task, P_j = period of high priority task, C_j= Execution time of priority task, hp(i)=the set of tasks which have a higher priority than task i.

W_1^0=15, W_2^0=5,W_2^1=8,W_2^2=9.8,W_3^0=2,W_3^1=4,W_3^2=5…..etc[24].

The cheddar schedulable table, theoretical scheduling and the simulation scheduling for the SWS/AIC system satisfies the requirements and can be used as a proof for certification.

5 Mapping of Kernel Architecture Analysis to Certification Artifacts as per RTCA DO-178C Workflow

Architecture analysis not only helps in understanding the requirements better and uncovering the design flaws earlier in the process but it also helps in the addressing the certification issues. RTCA DO-178C[16][17] is the civil aerospace software engineering guideline. The ADDL analysis report can be used to address the objectives provided in the 178C guidelines. The objectives addressed by this analysis are:

- *Software architecture is compatible with high level requirements.*
- *Software architecture is consistent.*
- *Software architecture is compatible with target computer.*
- *Software architecture is verifiable.*
- *Software architecture is conforms to standards.*
- *Software partitioning integrity is confirmed.*

In our case study, the software for SWS/AIC system is designed, developed and qualified as per the highest criticality level of the 178C guidelines. With the AADL approach for the kernel architecture, 6 objectives of the 71 objectives are addressed. Out of 71objectives, 6 are related to software architecture, in which 3 are independent. Table 1 shows the software criticality and the architecture level objectives.

Table 1 The objectives related to software architecture based on criticality level.

Criticality Level	Failure condition	Objectives	With independence
A	Catastrophic	6/71	3/33
B	Hazardous	6/69	0/21
C	Major	4/62	0/8
D	Minor	0/26	0/5
E	No Safety Effect	0/0	0/0

The analysis results by using AADL Inspector can be mapped to satisfy the objectives as shown in Table 1. Analysis report generated by the tool can be used as an artifact to verify the effectiveness of the scheduler as per the functional, operational and safety requirements.

6 Conclusion and Future Scope

The work focuses on the importance of RTOS architecture analysis in case of safety critical system and mapping the analysis to RTCA DO-178C compliance. AADL based architecture analysis helps in analyzing the RTOS architecture in terms of model statistics, resource allocation with respect to processor/s and threads, latency analysis & scheduling analysis of various processes and threads. This is demonstrated using the case study of the proven SWS/AIC kernel architecture. AADL-based approach proposed helps in developing the confidence of the kernel architecture for its functionality, operation and safety. The analysis outcome provides evidence which can be mapped to the certification objectives of RTCA DO-178C. Hence incorporating the AADL based approach in the engineering process make the process effective as it will uncover the requirements and design flaws earlier in the life cycle and increase the safety feature of the software.

Future work involves implementation of this approach for certification of highly complex RTOS used in the advanced avionics systems, such as Integrated Modular Avionics which uses the hard-real time space and times partition operating system.

Acknowledgments Authors would like to thank the Director, CSIR-NAL for providing them the opportunity to work in this area.

References

1. Bieber, P., Boniol, F., Boyer, M., Noulard, E., Pagetti, C.: New Challenges for Future Avionic Architectures (4), May 2012. pp10
2. Dobrica, L., Niemelä, E.: A Survey on Software Architecture Analysis Method. Ieee Transactions on Software Engineering **28**(7), July 2002. pp10
3. Dhage, S.: Qualification of RTOS for safety critical systems using formal methods. INDIAcom 2015. pp12
4. Singhoff, F., Legrand, J., Nana, L.: Scheduling and memory requirements analysis with AADL. In: International Conference Proceedings, November 13–17, 2005. pp11
5. Designing Safety-Critical Avionics Software Using Open Standards. http://www.google.com.unpublished
6. Howard, C.E.: Safety- and security-critical avionics software, February 1, 2011. http://www.militaryaerospace.com/articles/print/volume-22/issue-2/technology-focus/safety-and-security-critical-avionics-software.html
7. Donini, R., Marrone, S., Mazzocca, N., Orazzo, A., Papa, D., Venticinque, S.: Testing complex safety- critical systems in SOA Context, November 12, 2007. pp8
8. Alexander, R., Alexander-Bown, R., Kelly, T.: Engineering Safety-Critical Complex Systems. http://www.cs.york.ac.uk/nature/tuna/outputs/finalreport.pdf, pp27
9. Correa, T., Becker, L.B., Farines, J.-M.: Supporting the design of safety critical systems using AADL. In: 2010 15th IEEE International Conference on Engineering of Complex Computer Systems, pp. 331–336. pp6
10. Knight, J.C.: Safety critical systems: challenges and directions. In: Proceedings of the 24rd International Conference on ICSE 2002 (2002). ieeexplore.ieee.org, pp4
11. Nordhoff, S.: DO-178C/ED-12C the new software standards for avionics industry: goal, changes and challenges. http://www.sqs.com, pp26
12. Feiler, P.H., Gluch, D.P., Hudak, J.J., Lewis, B.A.: Embedded System Architecture Analysis Using SAE AADL, June 2004. Technical Note, CMU/SEI-2004-TN-005, pp45
13. Adalog, J.-P.R., Axlog, J.-F.T.: AADL Workshop, October 17–18, 2005. 2005 Overview of AADL Syntax
14. Casteres, J., Ramaherirariny, T.: Aircraft integration real-time simulator modeling with AADL for architecture tradeoffs, pp. 346–351 (2009). pp6
15. Rammig, F., Ditze, M., Janacik, P., Heimfarth, T., Kerstan, T., Oberthuer, S., Stahl, K.: Basic Concepts of Real Time Operating systems. Hardware-Dependent Software, Springer Science + Business Media B.V., 16–44 (2009). pp28
16. RTCA DO-178C Software Consideration in Airborne Systems and Equipment Certification
17. RTCA DO-333 Formal Method Supplement to DO-178C and DO-278A
18. Formal Methods for Software Architectures: Software Architectures, SFM 2003, Bertinoro, Italy, September 22–27, 2003. http://www.springer.com
19. Cofer, D.: "DO-178C", High Confidence Software & Systems Conference, May 8, 2012. pp33
20. Wang, Y., Ngolah, C.F.: Formal Description of a Real-Time Operating System using RTPA, vol. 2, pp. 1247–1250, May 4–7, 2003. pp3
21. CSIR-NAL: Software Design Description (SWDD) of SARAS aircraft
22. Noll, T.: Safety, dependability and performance analysis of aerospace systems. In: Third International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS 2014) November 1–5, 2014
23. Fisher, K.: Using Formal Methods to Enable More Secure Vehicles: DARPA's HACMS Program, September 16, 2014
24. Hugues, J., Singhoff, F.: AADLv2: an Architecture Description Language for the Analysis and Generation of Embedded Systems. ISAE, France