Defence Science Journal, Vol. 63, No. 2, March 2013, pp. 223-229


REVIEW PAPER

# ARINC 653 API and its application – An insight into Avionics System Case Study

Ananda C.M.*, Sabitha Nair, and Mainak G.H.

*CSIR-National Aerospace Laboratories, Bangalore–560 017, India*
**E-mail: ananda_cm@nal.res.in*

**ABSTRACT**

Traditionally automated systems in aircraft were realised using well defined functions that are implemented as federated functional units. Each functional units possesses its own resources with fault containment compared to multiple functions in single processing node. Integrated architectures are structured over the avionics cabinets or processing cabinets which house the hardware modules and software application partitions along with system software. These integrated modular avionics (IMA) applications supports distributed multiprocessor architecture. Both time and memory is shared among multiple avionics functionalities across the same platform with good protection mechanisms provided by ARINC 653. ARINC 653 is an additional layer of protection being embedded as part of real time operating systems supporting the partitioning protections using well defined application executive, and application programming interfaces (API). IMA uses set of partitions, which are scheduled across a major frame M consisting set of partitions $P_{tn}$ and each partition having set of task/process $\tau_n/P_{sn}$. The number of partitions and number of processes in each partition is a trade-off between the real time requirements and the resource. The paper also presents in brief, the API functionalities, its components, implementation, required interfaces, restrictions based on criticality of the avionics application. Error detection, control mechanisms for data integrity and validity for reconfiguration is also addressed. The experimental and simulation studies related to the API utilization as part of case study is addressed with four partitioned case study demonstrating the normal and failure scenario.

**Keywords:** Integrated modular avionics, application programming interfaces, memory partition, application executive

## 1. INTRODUCTION

The avionics architecture consisting of modules, sub-systems and systems in aerospace domain is complex hardware and software centric. Such architecture of federated era was no doubt very good in terms of fault containment, fault tolerance and was fool proof architecture. However, it has many disadvantages of having resources duplicated in each system in terms of processing, memory and peripheral systems. Advantages being excellent fault tolerant capabilities against the disadvantage of duplication of resources and complexity. Also such systems have disadvantages like increased weight, redundant computer resources in each line replaceable unit (LRU), higher looming volume, electrical interface complexity and physical maintenance.

Growth in the aerospace industry encouraged the avionics systems to utilise the increased processing power, communication bandwidth and hosting of multiple federated applications into a single integrated platform. This has been realized as integrated modular avionics (IMA). The IMA emerged as a platform for integrating multiple avionics severity applications on a common shared integrated computing environment[1,2]. Such system platform is realizable with well integrated ARINC 653 based real time operating system with time and memory partitioning with application executive (APEX) libraries[3].

IMA integration has the advantage of affordable low recurring expenses for software updates or upgrades without impacting the rest of the partitioned applications. Application partitions are well protected with time and memory protection mechanism provided by the underlying real time operating system (RTOS) ARINC 653 standard application executive.

In this paper it is attempted to bring a review of the APEX APIs and their usage. Also addresses the four partitioned application with data flow across all the four partitions.

## 2 ARINC 653 BASED RTOS APEX

### 2.1 Importance of Time and Memory Partitioning for Avionics Architectures

The ARINC 653 is one of the most critical specifications for the integrated modular avionicss definition[4], where protection and functional separation between applications are ensured with the use of partitions. This aims at enforcing fault containment, preventing fault propagation from one partition to another which eases the application's life cycle verification, validation and certification procedures. Also this enables to host multiple applications with varying DO 178B/C severity levels in different partitions.

Partitioning in IMA platform is quite critical and has strong impact on the avionics architectures like federated

Received 30 October 2012, revised 08 February 2013, online published 23 March 2013

and integrated. Traditionally avionics systems are more functionality centric with dedicated algorithmic and architectural methods for performance enhancement[5]. Avionics architectures are classified as two major sectors

(a) Federated architectures
(b) Integrated architectures.

Transiting from federated architecture to integrated architecture demands enormous effort with well defined domain understanding and platform support[6]. ARINC 653 time and memory partitioning mechanism as provided by supporting RTOS is very essential and mandatory. Partitioning uses appropriate hardware and software mechanism to restore strong fault containment to the maximum extent in such integrated architecture[7]. Partition can contain one or more application processes and each process having attributes like

- *Period:* Processes can be periodic or aperiodic. Periodic process has fixed period defined between successive releases of the process. Aperiodic processes are set with a unique value to indicate that they are not periodic and hence they do not have fixed period.
- *Time capacity:* Each process has fixed time for execution and has a deadline by which time the process should complete execution, which is a constant value.
- *Priority:* Each process has fixed priorities. This is the base and current priority of the process, based on the selection. The priority of the processes is pre-fixed as part of static scheduling.
- *State:* Process state can be dormant, ready, waiting or running based on the actual execution condition.

Process state can be dormant, ready, waiting or running based on the actual execution condition. With the above discussions, it is mandatory that the partitioning in IMA is critical and hence there is a guideline for partitioning if not a contract rule. Jon Rushby[7,8] ddefines and details spatial and temporal partition as;

- Spatial partitioning must ensure that software in one partition cannot change the software or private data of another partition (either in memory or in transit), nor command the private devices or actuators of other partitions.
- Temporal partitioning must ensure that the service received from shared resources by the software in one partition cannot be affected by the software in another partition. This includes the performance of the resource concerned, as well as the rate, latency, jitter, and duration of scheduled access to it.

With this partitioning system, the architectural difference in terms of complexity, modularity, inter operability and most important the recurring reduced efforts for certification is highly motivating factor for aerospace avionics system design and development.

Typical avionics architectures with and without partitioning if realized looks like as shown in Figs. 1 and 2 for federated and Integrated respectively. Hence, the partitioning system has tremendous potential in realization of efficient and cost effective avionics suite for civil aircraft requirements and in particular to gain advantage of recurring efforts for application design, and development.

The avionics application software is designed, developed and certified to DO 178B9 with varying severity in each partition as required based on the failure hazard analysis (FHA), failure mode effect analysis (FMEA), and system safety analysis (SSA)[10].

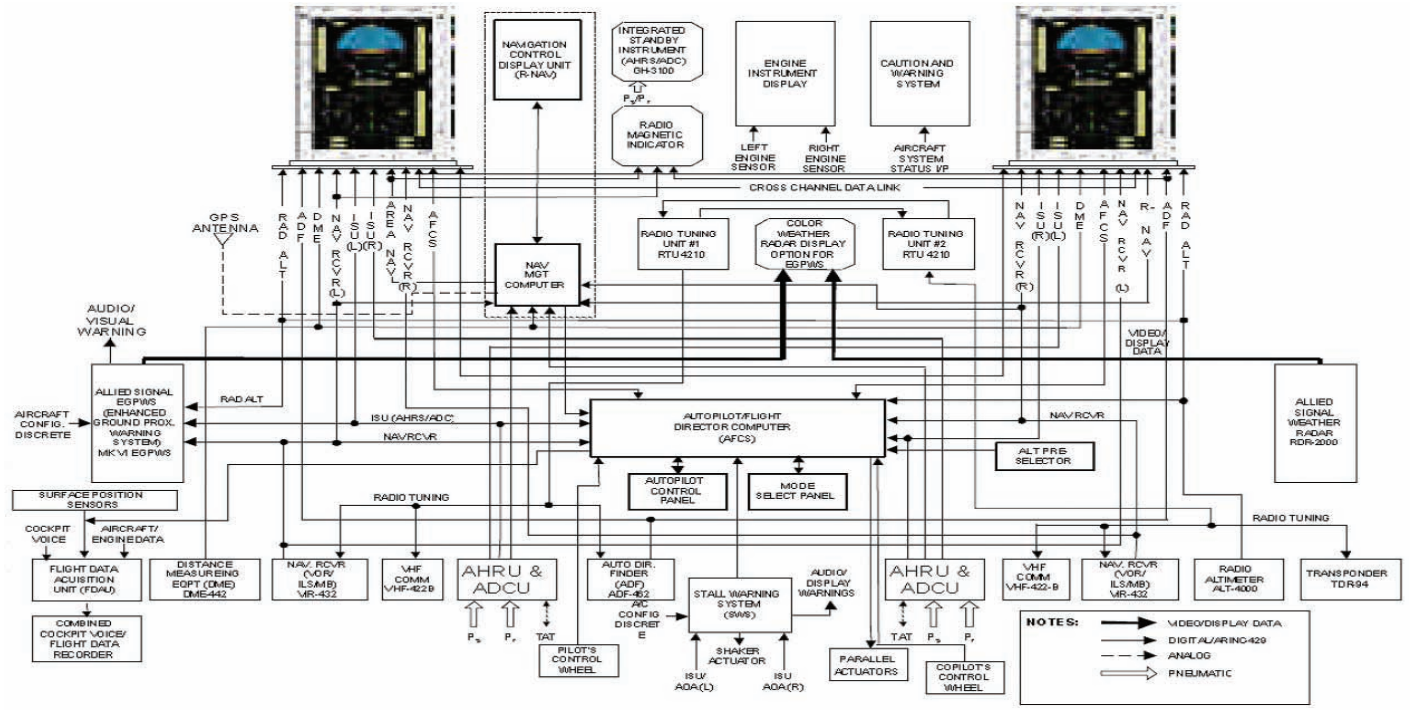


**Figure 1. Typical avionics federated architecture.**

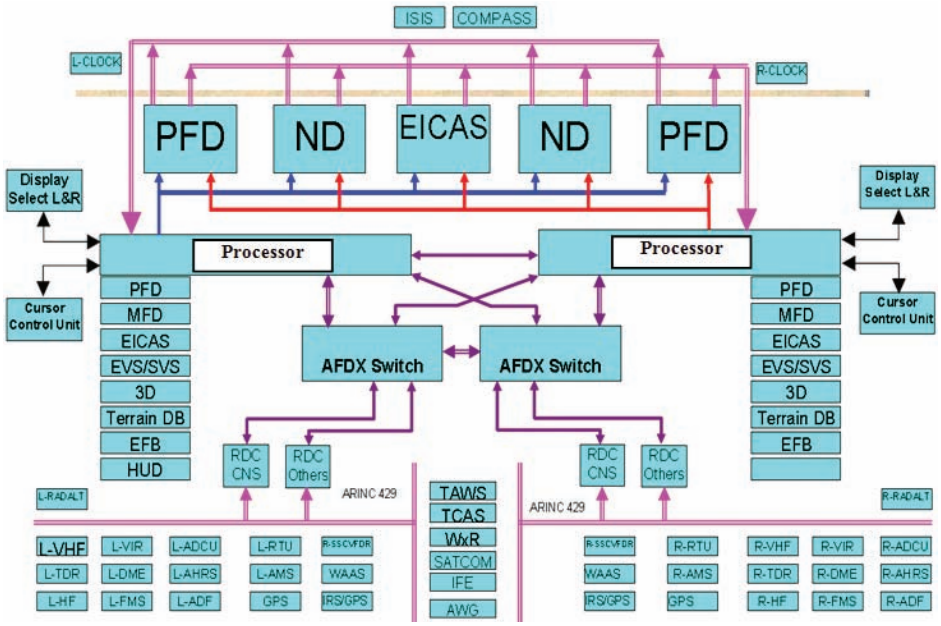


**Figure 2. Typical Integrated architecture based avionics suite.**

## 3. ARINC 653 PLATFORM

Typical avionics systems are scheduled with major frames. Each major frame has set of partition. Each partition is grouped in to a set of processes, each having modular sub-functionalities. As software components, each process consists of set of low level functions called the tasks as depicted in Fig. 3 and as in Eqn (1).

Figure 3 shows the set of partitions P, which are scheduled across a major frame M consisting of a set of partitions and each partition having a set of tasks/process. In some applications, task and process are interchangeably used, but it is better to use them separately under the multi-process operating system.

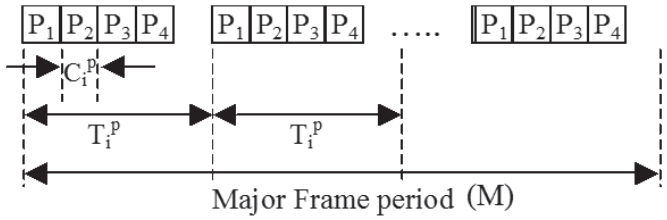


**Figure 3. Typical IMA partitioned schedule table with frames, process and tasks.**

Consider a major frame *M* having a set of partitions $P_{ti}..P_{tn}$ based on functionalities as in Eqn (1). Each partition $P_{ti}$ consists of a set of process $P_{si}..P_{sn}$ based on the applications sub-functionalities. Each $P_{si}$ consists set of tasks ($\tau_1$.. $\tau_2$). The number of partitions and number of processes in each partition is a trade-off to get the real time response based on capabilities of the hardware and software together. Normally the memory is statically allocated for each partition.

Each major frame (M)= $\{P_{t_1}, P_{t_2}, P_{t_3}, P_{t4}.....P_{t_n}\}$ (1)

Each partition ($P_t$) = $\{P_{s_1}, P_{s_2}, P_{s_3}, P_{s4}.....P_{s_n}\}$

Each task ($P_s$) = $\{\tau_1, \tau_2, \tau_3, \tau_4.....\tau_n\}$ ($\tau$ is single task)

Figure 4 shows the ARINC 653 multiple partitioned block diagram showing the typical blocks with layered interfaces from hardware up to the application. Application software hosted on each partition has one or more processes and these use the services provided by APEX using set of ARINC653 primitives. Apart from application partition, system partition also uses the services provided by the core software layer and may or may not use the

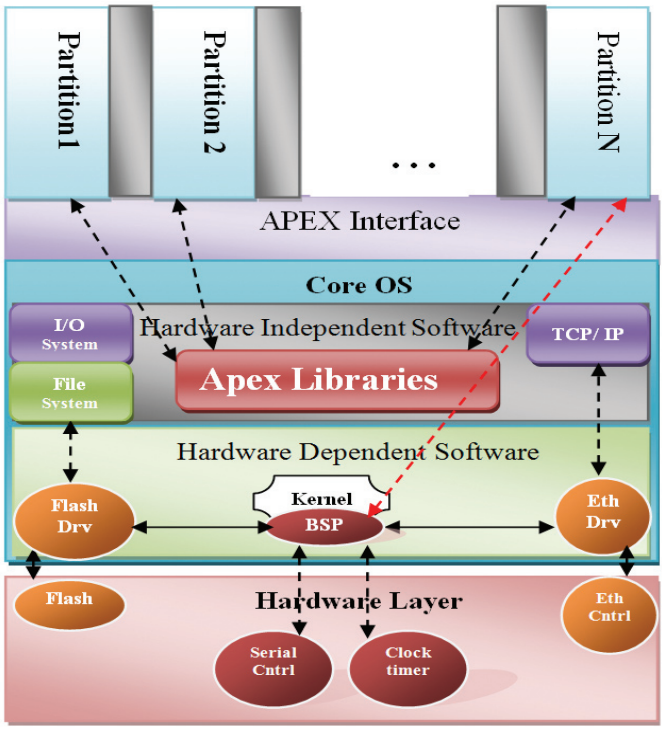


**Figure 4. Typical Multi partitioned ARINC 653 APEX platform.**

APEX interface[11]. Operating system Kernel handles the major services like process scheduling and management, time and clock management as well as inter-process synchronization and communication. Avionics functionality in a given partition is strictly governed by the time and memory partitioning mechanism and hence any faults in that partition is contained inside the partition. This is the strength and integrity of the ARINC 653 based platform for fault containment.

The application executive (APEX) interface services are mapped[12] into

- The RTOS kernel services like process management, time and clock management, inter-process synchronization and communication.
- The system specific operating system and processor/ platform functions are like hardware interfacing, device driver functions, facilities for application downloading and support for BIT.
- ARINC 653 functions like partition management and inter-partition communication.

### 3.1. Critical API's for Reconfiguration of Task or a Process

The APEX-Interface aims to provide services for: Partition management, process management, time management, intra-partition communication, inter-partition communication, and health monitoring. The complete list of APEX interfaces are detailed in the standard[13].

Intra-partition communication and Inter-partition communication in ARINC 653 platform is very critical and enables the secured data flow inside and across the partitions. Inter-partition communication uses sampling port services and queuing port services via messages. Intra-partition communication is for those communications and synchronization across processes present inside the partition. Intra-partition communication uses buffer services, blackboard services. The processes can be of aperiodic or periodic processes and is managed by APEX.

## 4. CASE STUDY WITH FOUR PARTITIONS

Attempt was made to experiment the time and memory partitioned mechanism for four partitioned system using VxWorks ARINC 653 environment.

The applications were implemented with task reconfiguration capability in the event of limited failure scenarios using reconfiguration algorithm[14,15]. Four partition systems have been implemented with sixteen tasks in each partition as shown in Fig. 5. These partitions are scheduled to execute at a frame time of 25 ms and are fixed priority static table driven schedule with time loading of approximately 50%. The execution time of each task in all four partitions was measured and is tabulated as shown in Table 3. Process attribute table is created for sixteen processes, and then processes are created with CREATE_PROCESS service and started with START service. Each partition is activated with NORMAL mode using SET_PARTITION_MODE service. Later these sixteen processes are called; time of execution of each process is captured using GET_TIME service.

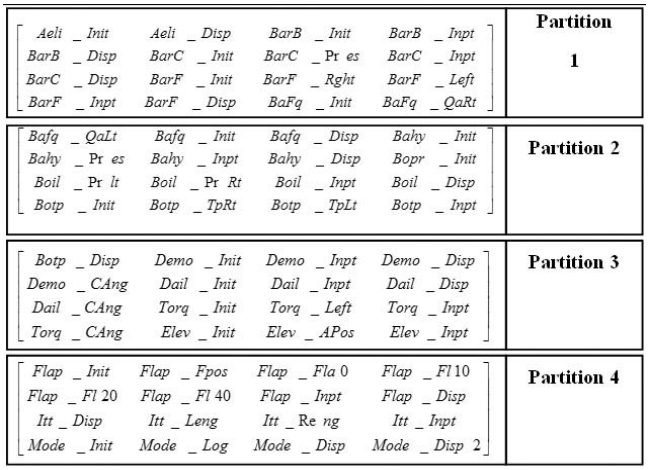


**Figure 5. Four partitioned avionics function.**

The CREATE_PROCESS service is used to create a process with defined attributes and allocate resources for it. Each process is created only once during the life of the partition. Also, all the processes in a partition must be defined in such a way that the necessary memory resources for each process can be determined at system build time. The field names in a PROCESS_ATTRIBUTE_TYPE structure are very critical and are an argument to the CREATE_PROCESS service. Field names are specified in Table 1. The current priority of a process can by changed dynamically through the SET_PRIORITY service.

**Table 1. Attributes of PROCESS_ATTRIBUTE_TYPE structure**

| | |
|---|---|
| NAME | String identifier for the process. It must be unique within the partition. |
| ENTRY_POINT | Starting address of the process. |
| STACK_SIZE | Size (in bytes) of the stack allocated to the process. |
| BASE_PRIORITY | Process initial priority. |
| PERIOD | Delay between two activations |
| TIME_CAPACITY | The elapsed time within which the process should complete its execution. |
| DEADLINE | Type of deadline (SOFT, HARD, or no deadline). |

APEX partitions within an ARINC 653 module communicate with each other using messages with the help of ports and channels. Communication was established in queuing mode with name, message size, message range, port direction, queuing discipline and port id using CREATE_QUEUING_PORT routine. Subsequently the messages were sent in this specified queuing port using SEND_QUEUING_MESSAGE service. When a partition is created corresponding partition XML configuration file, the application module and Core OS XML configuration file are created. Memory for each application is specified in application xml file which also consists of Entry Point, Initialization time and period. Here, the port should also be specified by the developer implicitly. The port element in the configuration file will specify whether the port is queuing or sampling along with the name,

size, queuing length and protocol. Xml files for coreOS, Applications, system partitions, schedules, connections and health monitor tables are referred in the module xml configuration file along with scheduling of partitions, in short it deals with the entire configuration of the project. It is possible to assign major and minor frame for all partitions. In order to send messages through a port, the source partition and the destination partition are to be mentioned along with the respective port name. This is in correspondence with the channel ID of module configuration file.

Deadline time is the absolute time by which the process should be complete. It starts as the return value of the GET_TIME service (current system time) plus the TIME_CAPACITY that is specified when the process is created. Threads periodically evaluates deadline time to determine whether the process is satisfactorily completing its processing within the allotted time (time capacity). Deadline time can be increased by issuing the REPLENISH service. Various services for monitoring are

- Status of the process is monitored using GET_PROCESS_STATUS service
- Process ID of calling process is read using GET_MY_ID service.
- Process ID any process with specified process name is read using GET_PROCESS_ID service.
- Process is started using START service and this service will cause process to enter the READY state.

In case study experimentation, each of the four partitions with processes were enabled and started with START service. The sequence of startup and execution is captured and is shown in Fig. 6 for partition 1. Similarly startup sequence has been verified for partition 2, partition 3, and partition 4, respectively.

The scaled time loading measurements carried out using the System Time Viewer of VxWorks Workbench is listed below for each partition. The real-time data for each of the tasks in all four partitions were captured using the debug ports of the real-time System-Viewer[16].

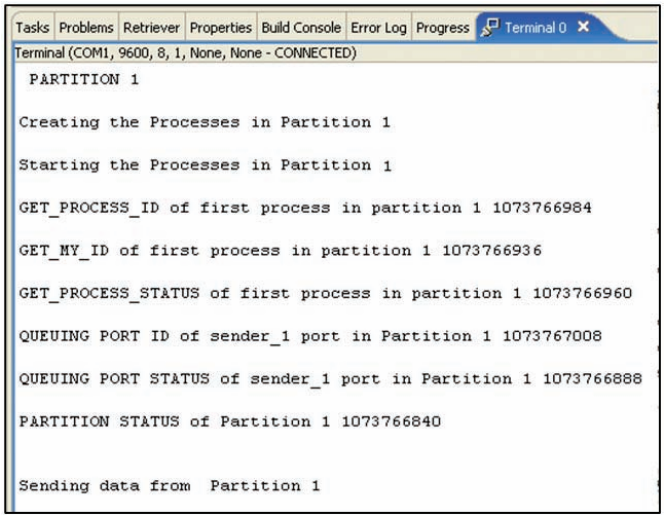


**Figure 6. Partition 1 startup.**

**Table 2. Execution time of all tasks in each of the four partitions**

Partition 1

$$\begin{bmatrix} 1.266 & 0.318 & 3.284 & 0.745 \\ 0.995 & 0.773 & 2.395 & 0.078 \\ 0.089 & 0.700 & 0.526 & 0.907 \\ 0.111 & 0.801 & 0.675 & 0.343 \end{bmatrix}$$

Partition 2

$$\begin{bmatrix} 1.266 & 0.318 & 3.284 & 0.745 \\ 0.995 & 0.773 & 2.395 & 0.078 \\ 0.089 & 0.700 & 0.526 & 0.907 \\ 0.111 & 0.801 & 0.675 & 0.343 \end{bmatrix}$$

Partition 3

$$\begin{bmatrix} 1.003 & 0.864 & 0.098 & 0.588 \\ 2.806 & 1.033 & 1.085 & 0.095 \\ 0.097 & 0.0945 & 1.273 & 0.544 \\ 2.576 & 0.451 & 0.654 & 0.483 \end{bmatrix}$$

Partition 4

$$\begin{bmatrix} 1.324 & 0.636 & 0.119 & 2.050 \\ 0.093 & 0.950 & 0.850 & 0.106 \\ 0.549 & 1.068 & 4.482 & 0.468 \\ 0.760 & 0.094 & 0.105 & 0.409 \end{bmatrix}$$

The time loading of each partition are 14.006 ms, 10.033 ms, 12.422 ms, and 11.692 ms respectively for partition 1, partition 2, partition 3, and partition 4 as shown in Table 3.

**Table 3. Time loading for each partition of schedule 1 and schedule 2**

| Schedule | Partitions | Allotted time (ms) | Used time (ms) |
|---|---|---|---|
| 0 | Partition 1 | 25 | 14.006 |
| | Partition 2 | 25 | 10.033 |
| | Partition 3 | 25 | 12.422 |
| | Partition 4 | 25 | 11.692 |
| 1 | Partition 1 | 25 | 14.204 |
| | Partition 2 | 25 | 11.853 |
| | Partition 3 | 25 | 12.239 |
| | Partition 4 | 25 | 11.383 |

Figure 7 shows the pictorial representation of the four partitions as case study with each partition's time loading measurements and hence the total time for each schedule. Consider a four partitioned system which executes in two scheduled times. Each partition consists of sixteen processes. In schedule 0 first four partitions will execute and in schedule 1 next four partitions will execute. Each partition is assigned with a partition window of 25 ms as shown in Table 3 and hence major frame time is 100 ms.

There are two types of inter partition communication services: Sampling port services and queuing port services. In queuing mode each message contains different data.

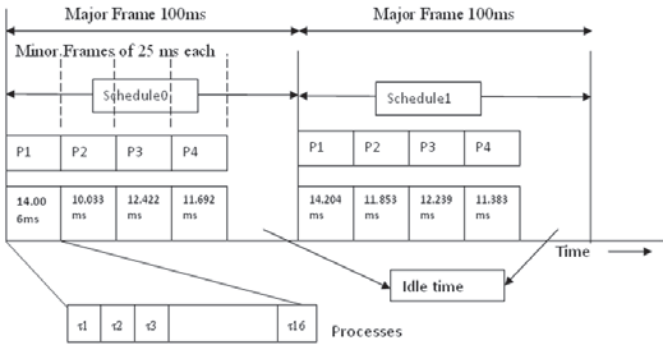


**Figure 7. Time loading of partitions for two schedules.**

Messages are queued and overwriting is not allowed so no data's are lost.

Messages will be there in source port until they are send, and will be stored in destination port until its read. Consider a two partitioned system with one process each. Queuing ports are created in each partition with CREATE_QUEUING_PORT service. Process1 in partition1 will send data to queuing port using SEND_QUEUING_MESSAGE service. Process 1 of Partition 2 will receive the data from queuing port using RECEIVE_QUEUING_MESSAGE service. GET_QUEUING_PORT_ID and GET_QUEUING_PORT_STATUS will return ID and Status of the ports respectively. Fig. 8 shows the capture of queuing port interfaces using the IDE toolset.

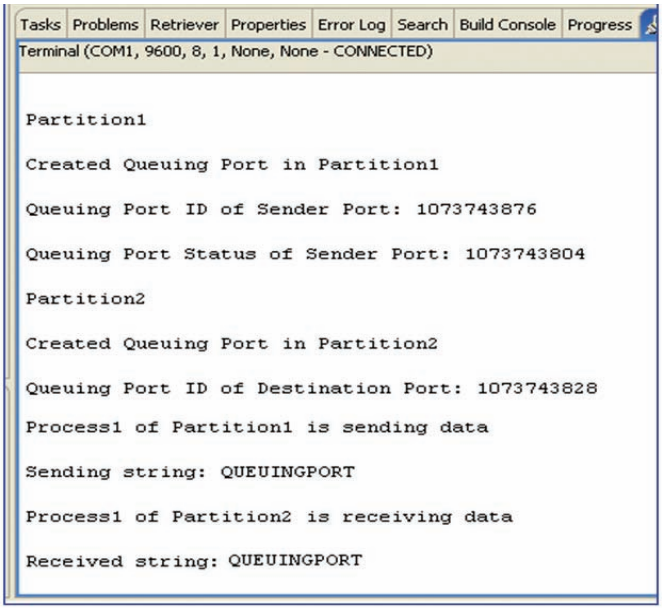


**Figure 8. Queuing port implementation in the case study.**

The System-Viewer is configured to display and capture the required execution parameters on the terminal window for real time monitoring. The result of time loading for each of the four partitions proposed algorithm functionality for different test cases is also captured using system viewer. The case study also addresses the failure test cases based on FHA/FMEA analysis.

Following is one sequence of failure test cases experimented and the results were captured using system viewer tool of VxWorks workbench.

- Test Case 1: All four partitions functioning normally with no fault
- Test Case 2: Failure of task 12 in partition1 and successful reconfiguration
- Test Case 3: Failure of task 02 in partition2 and successful reconfiguration
- Test Case 4: Failure of task 09 in partition3 and unsuccessful reconfiguration
- Test Case 5: Failure of task 14 in partition4 and successful reconfiguration

Fig. 9 shows the Test Case 2 failure simulation and hence the reconfiguration of the task captured using real

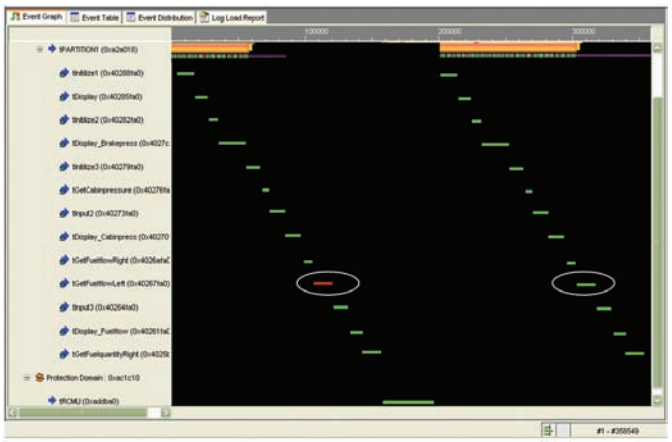

**Figure 9. Typical partition execution with a task failure (Test Case 2).**

time system viewer. More details on the control metrics, their validation and reconfiguration scheme along with the control metrics are available in Ananda[17].

### 4.1 Control Metrics for Reconfiguration

The safety feature of the above implementation is based on four control metrics for efficient functioning of the applications in each partition. They are

a. Reconfigurability information factor (RI)
b. Schedulability test/TL/UF (TL)
c. Context adaptability and suitability (CAS)
d. Context flight safety (CFS)

These control metrics helps the application for proper functioning in the event of any failure and its reversionary action if applicable for continued availability of the applications.

However, the data handled by these control metrics need to be clear from errors and hence error detection, correction and validation is the critical task for such implementation[17,18]. Case study also takes into account the various failure scenario of the system as shown Fig. 10.

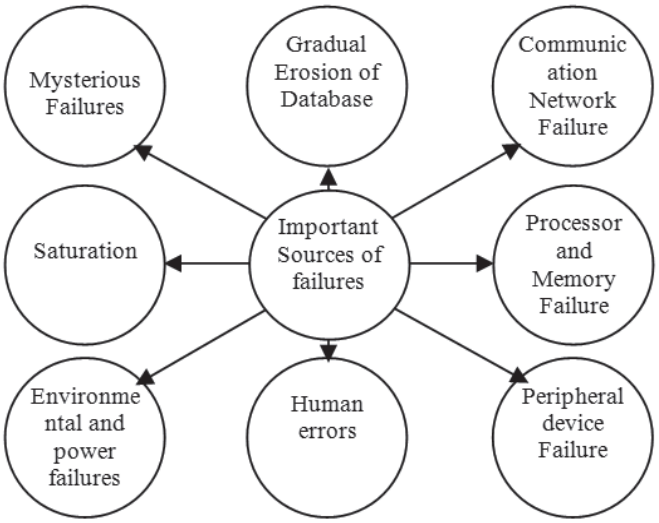


**Figure 10. Typical failure scenarios considered in the case study for various failure.**

## 5. CONCLUSION AND FUTURE WORK

Integrated modular avionics is well integrated with the ARINC 653 compliant RTOS including APEX Libraries. The APEX libraries play a very critical role in the safety and reliability issues of the system particularly for multi partitioned and multiple application requirements. ARINC 653 platform is the technology based for civil aircraft avionics and other systems for hosting multiple applications of different DO 178 severity levels on the same hardware resource. The set of APEX APIs provide handy and easy interface across the application and the underlying RTOS enabling quick implementation of applications with all the included capabilities of ARINC 653.

CSIR NAL is working on programs with ARINC 653 for avionics suite implementation for typical transport category FAR 25 class of aircraft. The result of this implementation is yet to be collected and is in the process of integration tests.

## ACKNOWLEDGMENT

Authors thanks Mr Shyam Chetty, Director NAL, Dr MK Sridhar, Adv (M&A) and Mr KG Venkatanarayana, Head ALD for their moral and management support. Authors acknowledge the interactive support from ALD Avionics and Wind River technical support teams.

## REFERENCES

1. ARINC Specification 653-1. Avionics application standard interface. Aeronautical Radio Inc Software, October 2003.
2. ARINC Specification 651: Design guidance for integrated modular avionics. Aeronautical Radio, Inc, Annapolis, MD, November 1991. Prepared by the Airlines Electronic Engineering Committee.
3. Audsley, Neil & Wellings, Andy. Analyzing APEX applications. *In* the Proceedings of IEEE Real Time Systems Symposium RTSS, 1052-8725/96 1996 IEEE, pp 39-44.
4. D0 297: Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations.
5. Russell W. Duren, Waco, Algorithmic and architectural methods for performance enhancement of avionics systems. *In* the 28th Digital Avionics Systems Conference, DASC 2009, pp. 1.D.4-1 – 1.D.1-6.
6. Watkins, C. & Walter, R. Transitioning from federated avionics architectures to integrated modular avionics. *In* the Proceedings of the 26th IEEE/AIAA Digital Avionics Systems Conference (DASC 2007), Dallas, TX, USA, Oct. 2007. pp. 2.A.1–1–2.A.1–10.
7. Rushby, John. Partitioning in avionics architectures: Requirements, mechanisms and assurance. Computer Science Laboratory, SRI International, FAA Technical Center and NASA Langley Research Center through contract NAS1-20334, and by DARPA and NSA through Air Force Rome Laboratory Contract F30602-96-C-0204. March 1999, pp. 1-68.
8. Rushby, John. Partitioning in avionics architectures: Requirements, mechanisms and assurance. SRI International, Menlo Park, California, NASA/CR-1999-209347, June 1999. pp-1-68.
9. Software considerations in airborne systems and equipment certification. RTCA/DO178B, RTCA Inc., Washington, DC, Dec 1992.
10. Analysis techniques for system reliability - Procedure for failure mode and effects analysis (FMEA). IEC 60812, 1985.
11. Rufino, José & Filipe, Sérgio. AIR Project Final Report, DI-FCUL, Report No.TR–07–35, Skysoft Portugal & FCUL, Portugal, December 2007.
12. VxWorks 653 programmers Guide 2.3. www.windriver.com/products/product.../PN_VE_653_Platform2_3_0410.pdf. (Accessed on 12/08/2012)
13. ARINC. ARINC Specification 653-2: Avionics application software standard interface, Part 1 - Required Services. Aeronautical Radio INC, Maryland, US. 2005.
14. Ananda, C.M. Re-configurable avionics architecture algorithm for embedded applications. *In* the IADIS International Conference on Intelligent System and Agents 2007, Lisbon Portugal, 2007, pp. 154-160.
15. Ananda, C.M. Improved availability and reliability using reconfiguration algorithm for task or process in a flight critical software. *Edited by* F.Saglietti and N Oster. Spriner LNCS publication, Springer-Verlag Berlin Heidelberg 2007, 2007, pp. 532-545.
16. Wind River Systems Inc, Wind River System Viewer user's guide, ver 4.7, 2005. http://www.windriver.com/products/product-notes/PN_WB_1110.pdf (Accessed on 12/08/2012)
17. Ananda, C.M. Reconfiguration of task in flight critical system - Error detection and control. *In* the 28th Digital Avionics Systems Conference (DASC) on Modernization of Avionics and ATM — Perspectives from the Air and Ground. The Florida Conference Center, Orlando, FL October 25 - 29, 2009, USA, @IEEE, pp. 1.A.4-1–1.A.4.-10.
18. Ananda, C.M. Validation of control parameters for a reconfigurable algorithm in a flight critical avionics application. *In* the International Conference on Aerospace Science and Technology (INCAST), INCAST-013, Bangalore India, 26-28 June 2008. pp. 310.1–310.6.

**Contributor**

**Shri C.M. Ananda** obtained his Masters degree in software systems from BITS, Pilani in 2003. He is currently pursuing his PhD in avionics. He is working at National Aerospace Laboratories on embedded systems, digital autopilot, engine instruments crew alerting systems, and stall warning system for SARAS aircraft. His areas of interest are: Integrated modular avionics, flight critical system, embedded distributed system, re-configurable avionics architectures and safety critical software.