WIND RIVER

white paper

Safety-Critical Software Development for Integrated Modular Avionics

Paul Parkinson
Senior Systems Architect, Wind River

Larry Kinnan
Senior Engineering Specialist, Aerospace & Defense, Wind River

Table of Contents

This technical paper presents recent trends in the development of safety-critical avionics systems. It discusses the emergence of Integrated Modular Avionics (IMA) architectures and standards and the resulting impact on the development of a commercial off-the-shelf (COTS) RTOS that is standards-compliant.

Introduction

Many avionics systems have been successfully developed using custom hardware and software. However, in recent years, the full life-cycle costs of customized systems have forced original equipment manufacturers (OEMs) to consider the use of COTS-based systems. At the same time, there has been a noticeable migration away from federated architectures, where each individual subsystem performs a dedicated function toward generic computing platforms that can be used in multiple types of applications and, in some cases, run multiple applications concurrently. This approach, known as Integrated Modular Avionics (IMA), results in fewer subsystems, reduced weight, less power consumption, and less platform redundancy. A number of civil and military research programs have sought to define IMA architectures, and while they differ in their approaches, they share the same high-level objectives:

- **Common processing subsystems:** These should allow multiple applications to share and reuse the same computing resources. This results in a reduced number of subsystems that need to be deployed and more efficient use of system resources, leaving space for future expansion.
- **Software abstraction:** This should isolate the application not only from the underlying bus architecture but also from the underlying hardware architecture. This enhances portability of applications between different platforms and also enables the introduction of new hardware to replace obsolete architectures.
- **Maximize reuse:** An IMA architecture should allow for reuse of legacy code. This reduces development time while affording the developer a method of redeploying existing applications without extensive modifications.
- **Cost of change:** An IMA architecture should reduce the cost of change since it facilitates reuse and lowers retest costs because it simplifies the impact analysis by decoupling the constituent pieces of the platform that execute on the same processor.

IMA also facilitates support for applications that have ever-increasing levels of functionality, including the interactions between complex applications, such as head-up displays, map display systems, and weather radar displays.

Although a number of IMA architectures and standards has emerged, the ACR Specification[1] and ARINC Specification 653[2] appear to have the widest adoption in the avionics community. The ACR Specification addresses architectural considerations, whereas ARINC Specification 653 defines at a high level an instance of a software implementation for an IMA architecture. These and other IMA standards place new demands on the software architecture, especially the RTOS implementation provided by the COTS supplier. Wind River has specifically addressed these needs by developing the VxWorks 653 Platform[3], which is being employed by the C-130 Avionics Modernization Program and 767 Tanker[4]. Boeing has chosen to use Wind River's VxWorks 653 Platform for the development of the Boeing 787 Dreamliner Common Core System (CCS)[5]. Other Wind River customers, including EADS[6], are using the platform to develop avionics systems and safety-critical applications.

The following sections consider the technical requirements for an integrated device software platform to support IMA applications and show how VxWorks 653 Platform (see Figure 1), fulfils these requirements—in particular within the context of ARINC 653 application development.

Application Development with Wind River's VxWorks 653 Platform

The ACR Specification defines two important concepts widely used in IMA: spatial partitioning and temporal partitioning.

Spatial Partitioning

Spatial partitioning defines the isolation requirements for multiple applications running concurrently on the same computing platform, also known as a *module*. In this model, applications running in an IMA partition must not be able to deprive each other of shared application resources or those provided by the RTOS kernel. This is most often achieved through the use of different virtual memory contexts enforced by the processor's memory management unit (MMU).

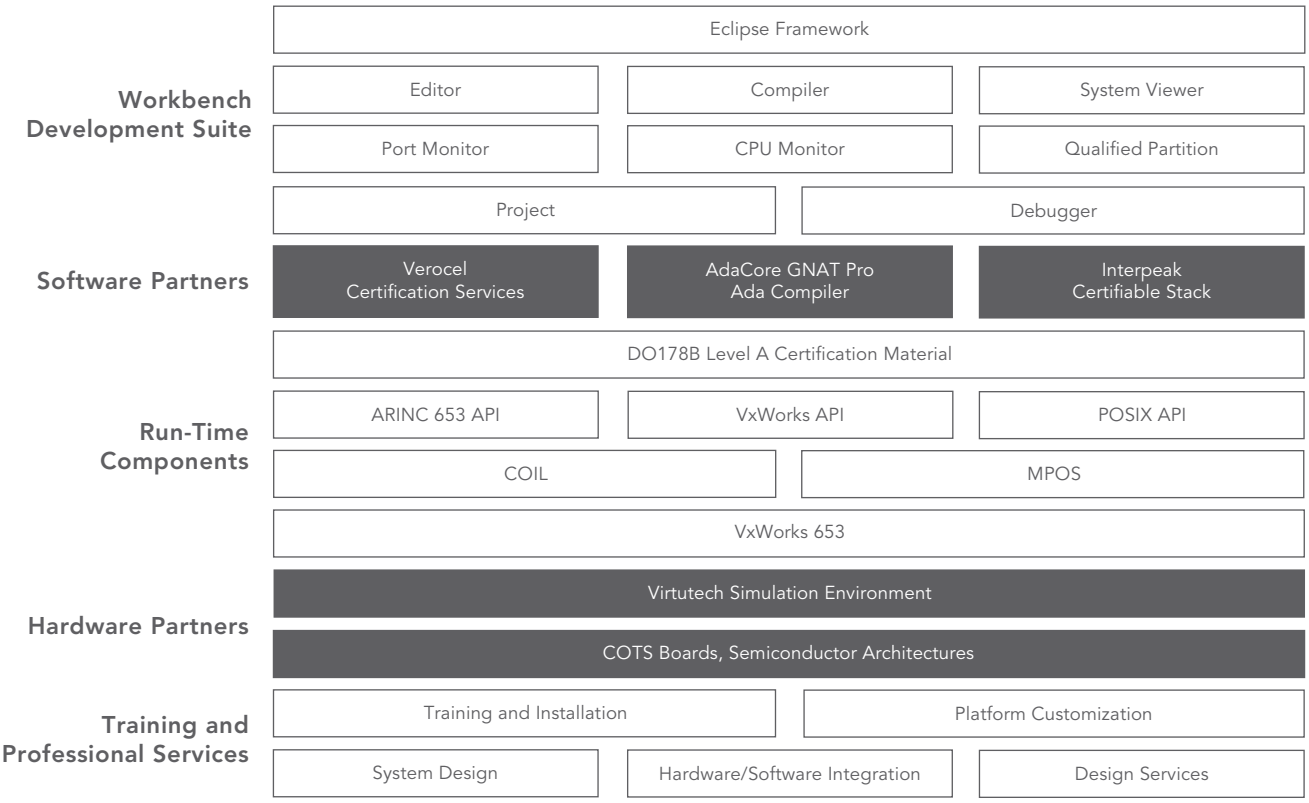

Figure 1: Wind River's VxWorks 653 Platform

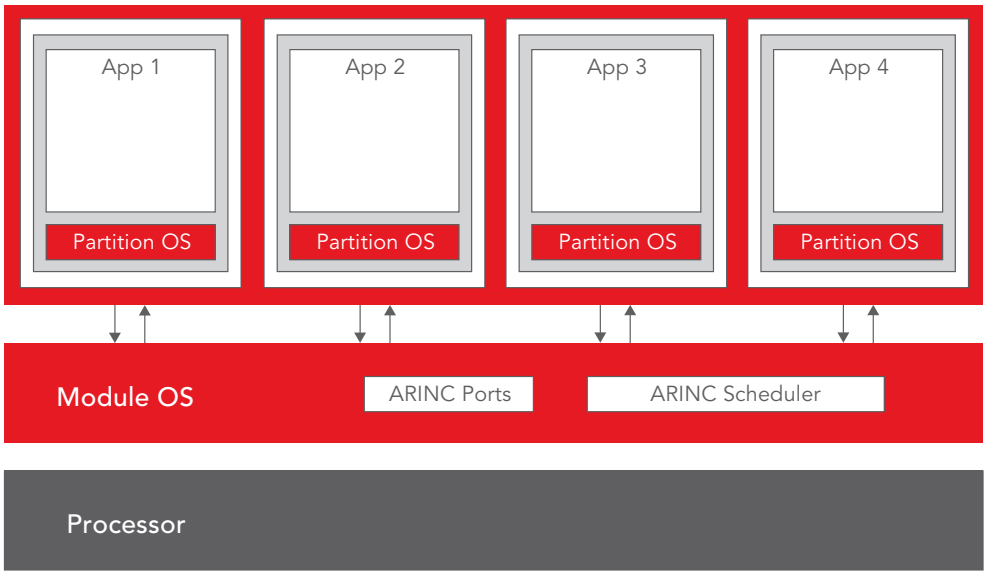

Figure 2: VxWorks 653 RTOS Architecture

These contexts are referred to as *partitions* in ARINC 653. Each partition contains an application with its own heap for dynamic memory allocation and a stack for the application's *processes* (the ARINC 653 term for a context of execution). These requirements affect the design and implementation of the RTOS kernel and language run-time system. For example, VxWorks 5.5 uses a shared virtual address space for applications and provides basic support through the MMU to prevent accidental or malicious access to program code by errant applications, without incurring the performance overhead of a full process model. VxWorks 6.x and VxWorks 653 provide an environment that uses the MMU to enforce separate contexts.

However, in an IMA environment, memory protection alone would not prevent an errant application running in a partition from consuming system resources, which might have a detrimental effect on an application running in another partition. This can have serious consequences where multiple applications of differing levels of criticality are running on the same processor. This problem cannot be resolved through the use of a full process model alone; instead it requires the development of an RTOS that specifically addresses the needs of IMA. The VxWorks 653 operating system was designed specifically for this purpose and supports the ARINC 653 model in the implementation of the kernel architecture (see Figure 2).

- The *module* OS interacts directly with the computing platform (core module), providing global resource management, scheduling, and health monitoring for each of the partitions. It also uses a board support package (BSP), the hardware-specific configuration required to run on different processors and hardware configurations.
- The *partition* OS is implemented using the VxWorks microkernel and provides scheduling and resource management within a partition. Communication with the module OS occurs through a private message-passing interface to ensure robustness. The partition OS also provides the ARINC 653 APEX (application/executive) interfaces for use by applications.

This architecture, which represents the virtual machine approach as described in "Partitioning in Avionics Architectures: Requirements, Mechanisms and Assurance"[7], provides a means of fulfilling the requirements of ARINC 653, while providing a flexible, extensible framework not easily achieved with a monolithic kernel implementation or UNIX-like implementations. Within the framework, individual partitions are implemented using memory-protected containers into which processes, objects, and resources can be placed, with partitioning enforced by the MMU (virtual machines). Each partition has its own stack and local heap, which cannot be usurped by applications running in other partitions. The partitions also prevent interference from errant memory accesses by applications running in other partitions.

Figure 2 shows the conceptual implementation of the VxWorks 653 architecture. The RTOS features the ability to have a single, shared partition OS library (shared read-only text) in order to simplify configuration, testing, and certification; it can also have separate, distinct configurations of the partition OS for one or more partitions (referred to as *MPOS*, or multiple partition operating system).

Temporal Partitioning

Temporal partitioning defines the isolation requirements for multiple applications running concurrently on the same computing platform. This ensures that one application may not utilize the processor for longer than intended to the detriment of the other applications. ARINC 653 addresses the problem by defining an implementation that uses partition-based scheduling. A partition is scheduled for a time slot of defined width, and other partitions may be allocated time slots of similar or differing durations. Within a time slot, a partition may use its own scheduling policy, but at the end of the time slot, the ARINC scheduler forces a context switch to the next partition in the schedule. This model is sufficiently flexible to enable existing federated applications or new IMA applications developed in isolation to be hosted on a core module. However, partition scheduling and verifying that boundaries and schedules are not violated, and taking appropriate corrective action, inevitably create additional complexity.

In VxWorks 653, the module OS performs ARINC 653 scheduling of the individual partitions. Within each time slot, the partition OS uses the VxWorks scheduler to perform preemptive, priority-based scheduling (see Figure 3). This means that all process level scheduling occurs within the partition space, enabling greater scalability and stability (minimal jitter) in the system even at high system clock rates (clock rates greater than 1 millisecond). It also fully implements the priority-ceiling protocol to prevent unbounded priority inversion (see the next section, "Priority Inversion, Priority Inheritance, and Priority Ceilings").

The implementation of VxWorks 653 is fully compliant with ARINC 653 Supplement 2, Part 1 (653-2).[8] It also supports optional mode-based scheduling, where up to 16 schedules can be predefined and used for different modes of flight or for staged initialization. Transitions between modes are achieved through a restricted API call, `arincSchedSet()`, and may be performed at the next major frame, next partition window, or next timer-tick boundary. The health monitoring system (HMS) validates the new schedule before being adopted (this is discussed later in the section "Health Monitoring System and Restarts").

ARINC 653 provides a highly deterministic scheduling methodology that may not be suitable for some applications, as it can lead to excess partition idle time or high clock-tick rates to insure high-rate sampling of data. To accommodate these types of applications, VxWorks 653 provides an option for priority preemptive scheduling (PPS) of partitions. This method permits slack stealing by allowing designated partitions to consume what would otherwise be idle time in the defined ARINC schedule.

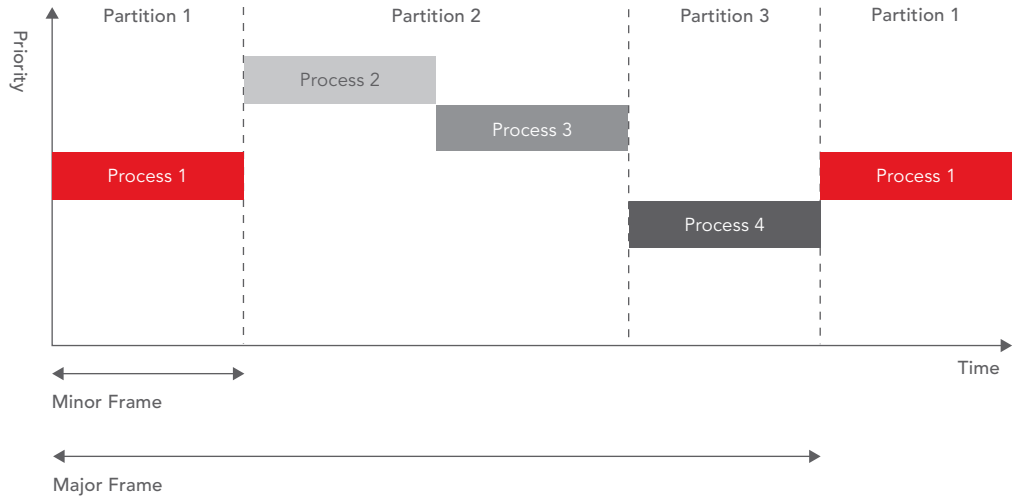

Figure 3: VxWorks 653 Temporal Partitioning

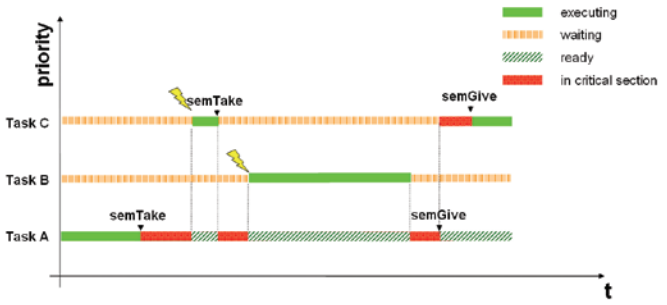

Figure 4: Priority Inversion Example

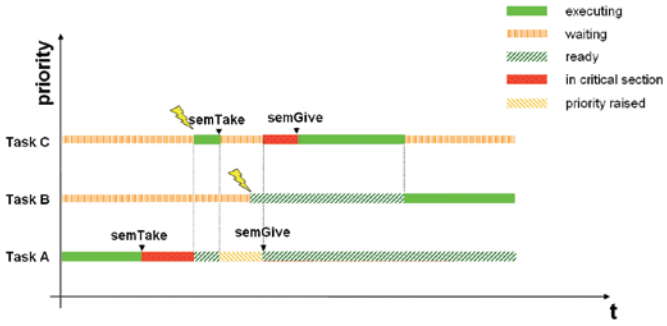

Figure 5: Priority Inheritance Example

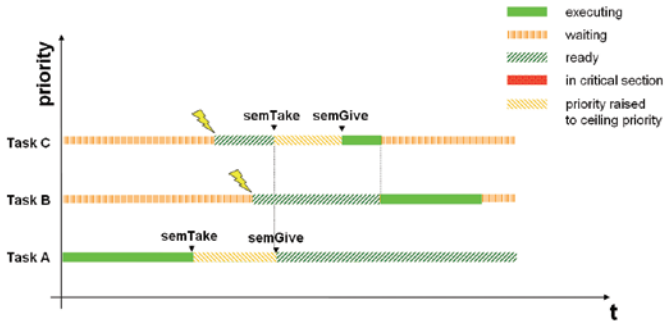

Figure 6: Priority Ceiling Example

Priority Inversion, Priority Inheritance, and Priority Ceilings

Priority inversion is a concern for device software designers. It occurs when a high-priority task is unable to run because a mutex (or binary semaphore) it wants to acquire is held by a low-priority task, which in turn is prevented from running by a medium-priority task (see Figure 4).

This problem is overcome in many RTOS implementations by the implementation of the *priority inheritance protocol*. This scheme associates a priority with each mutex, and the priority of a task holding the mutex is raised to the priority of the highest task requesting the mutex, as shown in Figure 5. Rate monotonic analysis (RMA) is supported when using priority inheritance or priority ceiling protocol, but additional analysis to determine worst-case execution time may be required when using priority inheritance because of possible chained blocking conditions.

The priority ceiling protocol is an alternative method used primarily to prevent chained blocking. Here, the mutex is initialized with a priority higher than any of the tasks that may acquire the mutex. When a task locks on this mutex, it is elevated to the priority ceiling (see Figure 6). While some RTOS vendors have used proprietary implementations for the priority ceiling protocol, Wind River uses the POSIX implementation[9] because it is an internationally recognized standard that enables legacy applications to be easily ported to IMA platforms running VxWorks 653 (see the section "Heterogeneous Application Support").

There is also a requirement to ensure that applications do not interfere with each other through use of kernel processing on behalf of the application. In ARINC 653, this is prevented through the use of a sophisticated model. Here, a partition layer is responsible for servicing APEX calls, and only those that require interaction with the module layer are passed over. The time spent servicing the kernel calls is scheduled as part of the partition, which prevents the application from stealing time from other partitions.

Wind River's VxWorks 653 extends this concept by providing the systems integrator at compile time with the ability to limit the number of concurrent blocking APEX calls. This is done through the static configuration of the number of kernel worker threads. (A worker thread performs kernel operations on behalf of an ARINC process making an APEX call). When all of the worker threads are busy servicing kernel API calls, the next APEX call requiring kernel interaction will result in the API call blocking until a worker thread becomes available. This is enforced by configuration data that defines allowable operations and resources on a partition basis. The configuration data is separate from the partition and its application, so changes can be isolated and rebuilt without altering the partition application itself. This significantly reduces the cost of change.

ARINC 653 Application Development

The ARINC 653 APEX, sometimes referred to as the ARINC 653 API, provides a general-purpose interface between the operating system and the application software. The ARINC 653 API also provides an abstraction layer that hides the implementation details of a particular ARINC 653–compliant RTOS from the application and the underlying architecture of the core module. This facilitates porting of the application to other ARINC 653 platforms, an important consideration for safety-critical IMA systems such as the flight computer, which requires dual-redundant or even, in the case of the Boeing 777, triple-redundant systems[10].

The ARINC 653 APEX also provides a model of static system configuration and initialization. Here, the number of ARINC processes is known ahead of time, and they are created in the partition through startup code using the `CREATE_PROCESS()` API. All other partition objects are created from the partition heap, and once this has been done, the partition is activated

through a call to `SET_PARTITION_MODE(NORMAL)`. At this point, the partition OS scheduler is activated and schedules the processes within the partition. Once the application has started, no further objects or processes may be created dynamically. This ensures a controlled, deterministic startup sequence for safety-critical applications and deterministic, fixed usage of resources.

ARINC 653 also provides excellent constructs for both intrapartition and interpartition communication. ARINC 653 blackboards and buffers assist with intrapartition communication. Blackboards provide a convenient write-once/read-many mechanism; buffers provide the ability to send and receive messages that are always stored in FIFO order but permit the receiving process to receive them in either FIFO or priority order. Additionally, semaphores and events can be used for synchronization.

ARINC 653 ports facilitate interpartition communication. The same naming scheme can be used for ports resident on the same processor or on another core module in the same IMA cabinet or in another IMA cabinet. This prevents applications from making architecture-dependent and/or configuration-dependent assumptions, aids portability, and eases reconfiguration by the systems integrator. The ARINC 653 ports facility provided by VxWorks 653 also allows the definition and use of a pseudoport, whereby an ARINC sampling or queuing port is connected to a module OS device driver to achieve intermodule communications while presenting a standard ARINC API to the application.

Heterogeneous Application Support

Although many IMA applications are developed from scratch, there is a wealth of existing applications on federated systems. These applications may be developed in different programming languages and use different scheduling models but may still need to communicate with each other in an IMA environment.

Wind River has addressed this need by providing support for heterogeneous applications running within individual ARINC partitions. This enables an Ada 95 application using the Ravenscar restricted-tasking profile to run on top of the VxWorks 653 partition OS (this is discussed at length by Parkinson & Gasperoni[11]). POSIX-compliant applications can run in a partition in a similar way, and communication between these applications can be implemented using ARINC 653 ports (see Figure 7).

System Configuration

The ARINC 653 architecture guarantees resource availability through the use of system and partition configuration records (also known as *system blueprints* in the IMA community). Their purpose is to enable the configuration of IMA applications developed by one or more OEMs onto a shared IMA platform that is configured by the systems integrator. The partition configuration records define the characteristics of each OEM application in terms of memory requirements, processor requirements, and ARINC port utilization. The system configuration record defines the capabilities and capacities of the IMA platform and references and validates individual partition configuration records. This scheme enables the systems integrator to ensure that the demands

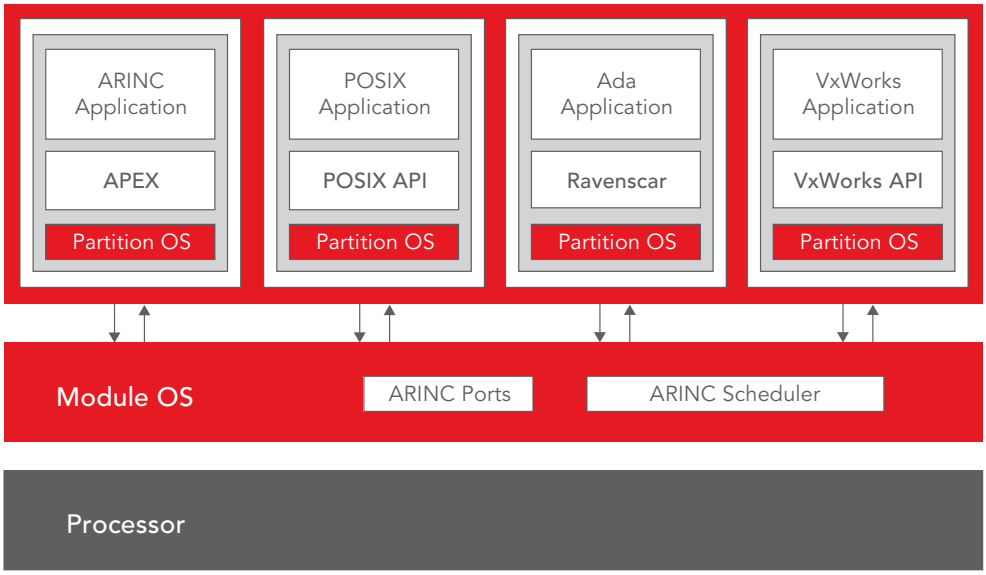

Figure 7: VxWorks 653 Heterogeneous Application Support

of the applications are consistent with the performance of the platform and that individual applications do not exceed their allocated resources.

The current revision of ARINC Specification 653 provides only a high-level definition of the structure and content of the configuration records and leaves the implementation up to the RTOS implementer, although a sample XML-based configuration is presented. VxWorks 653 uses the following process:

Step 1. At system initialization, the boot code loads the module OS and system and partition configuration records.

Step 2. The module OS initializes itself, starting its own subsystems.

Step 3. The module OS loads the application partitions and their applications.

This process decouples the configuration of the module OS binary image and the system and partition configuration records binary image from the partition applications. In this way, individual applications and subsystems can be developed separately then integrated easily on the target file system. Individual partition applications can also be upgraded in a straightforward manner without requiring changes to the module OS configuration. This results in significantly less recertification effort and greater flexibility for OEMs and systems integrators.

VxWorks 653 has extended the sample XML-based configuration to provide application developers, platform providers, and systems integrators with a complete and qualified toolset, coupled with data files to configure and initialize an IMA platform. This process, referred to as *independent build link and load* (IBLL), reduces the cost of change while providing a fully configurable run-time environment. The process also fully realizes the goals as stated in DO-297, Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations[12].

System and partition configurations can be changed without rebuilding the entire application or platform, which significantly reduces the impact-analysis burden for the system integrator when upgrading and modifying an existing system. Since the tools used to generate the configuration records in VxWorks 653 generate binary data directly from the XML configuration data, the tool is much simpler to use, and therefore more easily qualified than other implementations. Those often rely on more general-purpose mechanisms, such as a C compiler, to generate the binary configuration data for use by the system.

Health Monitoring System and Restarts

ARINC 653 defines the concept of a health monitor (HM) within an IMA system. The HM is responsible for "monitoring hardware, application, and operating system faults and failures," and it is the role of the HM to help "isolate faults and to prevent failures from propagating." Though the concept may appear straightforward, it is actually complex, requiring a sophisticated systemwide health monitor to track errors and perform reconfiguration and recovery. The response to an individual fault depends on the nature of the fault, its severity, and the error management policy defined by the systems integrator.

The VxWorks 653 HMS is a sophisticated framework that acts as an intrinsic part of the VxWorks 653 architecture. It fulfils all of the requirements of ARINC 653 and provides extensions relevant to systems integrators intending to use dynamic reconfiguration (in particular, mode-based scheduling). The design and implementation of the VxWorks 653 HMS is sufficiently rich that only an overview can be provided here.

The HMS architecture consists of systemwide HM server and HM agents (called *process-level handlers* in ARINC 653) residing in individual partitions, and the architecture also includes support for the module OS. The HMS processes events occurring in the system that need attention; these are known as *faults*, though they may represent either a negative or a positive event, such as a hardware exception or crossed threshold (a fault is represented in software by an *alarm*). The framework also supports *messages*, another type of event used for logging or other behavior configured by the systems integrator. Note that HMS use is not bound only to the ARINC support in VxWorks 653, giving added flexibility to the application developer.

The framework provides the ability to perform health monitoring at three levels—process HM, partition HM, and core module HM—through three types of services: alarm detection, alarm logging, and alarm response. The partition HM and module HM are table-driven and provide a mapping between a code and an appropriate handler. To aid portability, the framework uses ARINC 653 definitions for error codes, including events such as missed deadline, numeric error, illegal request, and power fail. XML-based configuration data is used to configure the framework and create the table-driven mapping for run-time usage by the system. Alarm response depends on the error level: module-level responses include reset or shutdown; partition-level responses include the restart of a partition.

At partition creation, a cold start is used to allocate and initialize partition objects; whereas a warm start, which reinitializes but does not allocate objects is used at partition reinitialization or restart. For process errors, the responses are application-driven; the action taken is dependent on the error type and its context. To facilitate warm restarts,

VxWorks 653 supports the use of persistent data types that provide for preservation of critical data during the warm start operation. This simplifies the operation and provides a mechanism to increase the speed of startup in such situations.

Tools for Safety-Critical Systems Development

Although RTOS run-time functionality is a major consideration, a discussion of IMA application development would be incomplete without reference to development and debugging tools. The quality of development and debugging tools can have a dramatic effect on development time scales. Tools designed for federated application development may not be suited to IMA development, as they need to support IMA models and scheduling modes.

Wind River's VxWorks 653 Platform provides an integrated development environment (IDE) with the Eclipse-based[13] Wind River Workbench development suite[14]. This state-of-the-art environment includes project configuration, code browsing and build, VxWorks 653 Simulator and target debugging, and the Wind River System Viewer analyzer. Figure 8 shows Workbench debugging an application running in an ARINC partition. In addition to capabilities provided by Wind River, Eclipse plug-ins for open source and partner tools can further extend and customize the environment.

The dynamic visualization capabilities are a real benefit to an application developer because they provide graphical feedback on the behavior of ARINC, POSIX, and VxWorks applications; interactions between partitions; and the operations of the HMS. Workbench can be used to browse, navigate, and comprehend Ada, ARINC, POSIX, and VxWorks applications. Wind River System Viewer is partition-aware, and can display ARINC processes, POSIX threads, and

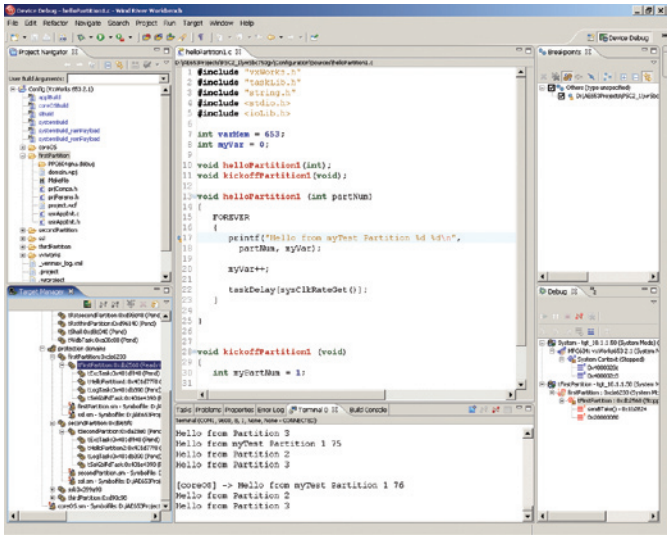

Figure 8: Wind River Workbench Showing VxWorks 653 Partition Debugging

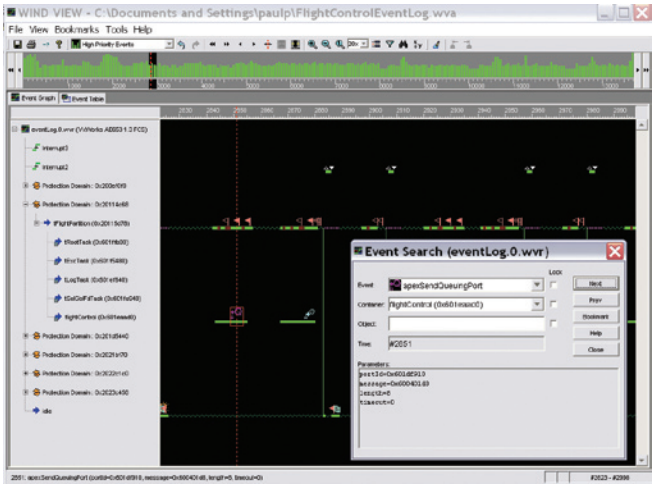

Figure 9: Wind River System Viewer Showing ARINC Partition Behavior

VxWorks tasks. It can be invaluable in displaying the internal behavior of an application, as shown by the interprocess communication between ARINC processes through ARINC queuing ports in Figure 9.

It is important for the ARINC application developer to not only visualize the behavior within an individual ARINC partition but to view application operation in the certifiable environment as well as interpartition communication through ARINC ports and channels. This is achieved through the use of CPU time usage monitoring, memory usage monitoring, and port-monitoring tools built into the RTOS and certified as part of the VxWorks 653 run-time system. These RTOS monitors are then coupled with DO-178B-qualified host tools for display and logging of the data in the test for credit environment. The monitors and tools provide unprecedented levels of insight into the operation of the system from development environment through final certified flight configuration.

Security Considerations for Networked IMA Systems

Security is now becoming a greater consideration in avionics systems (see discussion by Tingey & Parkinson[15]). Security at the aircraft level can be achieved through the use of firewalls that restrict interactions between different types of subsystems and separate flight systems from OEM systems and airline systems[16]. However, with the advent of IMA, there is a drive to increase—in a certifiable manner—the networking connectivity within these domains. This goal presents some interesting design challenges.

TCP/IP and related networking protocols require considerable effort to certify, and systems designers have to achieve a balance between functionality and the suitability for certification. In particular, DO-178B Level A certification of a full TCP/IP stack is rather onerous. Some have advocated the use of proprietary implementations utilizing a slave processor to implement the network stack for the master processor.

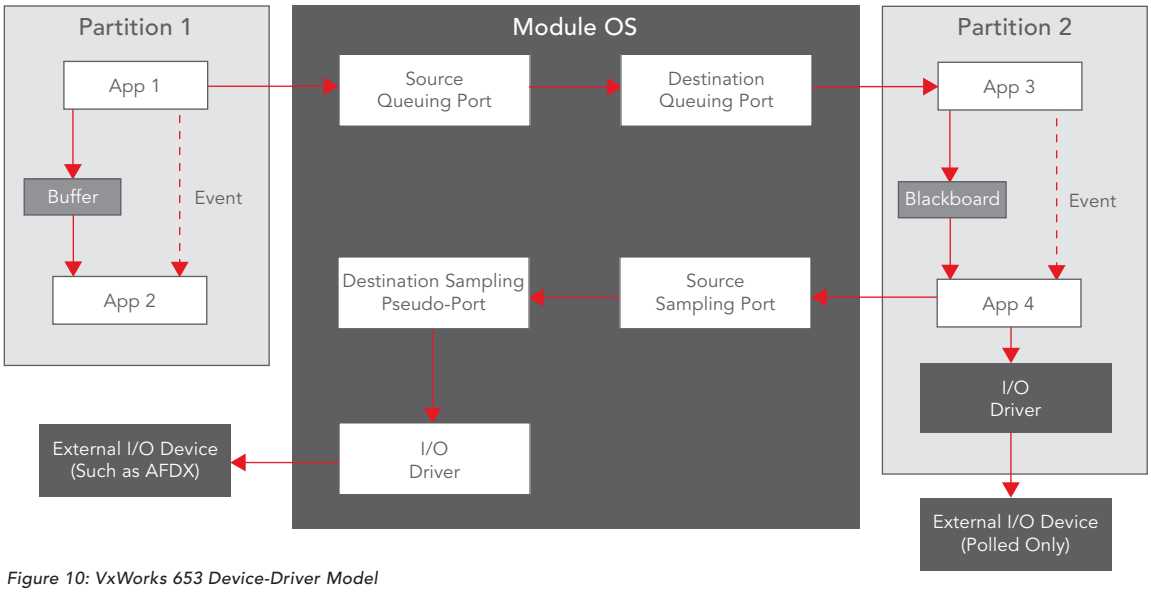

Figure 10: VxWorks 653 Device-Driver Model

However, this custom configuration uses additional hardware and restricts software portability. Since this defeats two of the goals of IMA outlined earlier, it can only be viewed as a retrograde step. Wind River's approach is to provide an optional certifiable network stack starting with VxWorks 653 version 2.2 which implements UDP/IPv4 capabilities and is designed for certification to DO-178B Level A. This provides a level of functionality most customers require while still providing for extensibility to add other protocol layers such as IPv6 at a later time.

There are also security implications at the core module level, which are worth highlighting. For example, application partitions run in the processor's user mode are unable to execute privileged processor instructions. In addition, if the partition OS is unable to service an APEX call on behalf of an application, it is passed to the module OS for validation prior to execution. Types of validation include: address validation for the memory range within view of the partition; boundary checks; module OS object access rights; and data structure integrity/consistency checking. VxWorks 653 also provides a scalable system-call privilege mechanism, whereby one partition can have more authority than others and provides a great foundation to meet the requirements of ISO-15408[17] (see the related white paper from Wind River[18]). There are also security restrictions in relation to the HMS; for example, only a privileged partition acting as a mode manager for the system may request an ARINC schedule change. Each of these techniques contributes to increased security.

Safety Considerations for IMA Systems

Although the certification of IMA systems is a relatively new endeavor, many aspects build on the methods used in the certification of existing federated systems to various certification standards[19 & 20]. For example, the concept of reusable software components with DO-178B certification evidence in a safety-critical application is well-documented[21] and has been applied successfully to VxWorks certification in a federated application on an FAA program[22].

The decoupling of the module OS and partitions in VxWorks 653 through the use of spatial partitioning enables this concept to be extended further. Now, a VxWorks 5.x application that has been previously certified to DO-178B Level C can be used in a separate partition on the same IMA platform as a new DO-178B Level A application, without the Level C application needing to be recertified to Level A. This technique can also be applied to I/O drivers and networking stacks (such as TCP/IP). These are placed into a separate VxWorks 653 I/O partition that is isolated from both the module OS and application partitions. Communication with application partitions is achieved through the use of ARINC ports, and interaction with the module OS is restricted to supervisor-mode driver routines. This prevents uncertified code from affecting the correct operation of the module OS or application partitions (see Figure 10).

Summary

The avionics industry is in the midst of major shift toward IMA, even though the continued evolution of IMA architectures and standards presents challenges for standards organizations, OEMs, and commercial vendors alike.

Wind River provides an integrated device software platform. Wind River's VxWorks 653 Platform brings together a standards-complaint COTS RTOS and all of the tools needed to successfully develop safety-critical IMA applications. The platform not only enhances developer productivity, it makes sure that the complexity and effort involved in certification does not impinge on developers.

In addition, the heterogeneous support for ARINC 653, Ada, POSIX, and VxWorks applications in an IMA environment facilitates maximum software reuse and porting of existing federated applications to VxWorks 653.

References

1 DO-255, "Requirements Specification for Avionics Computer Resource (ACR)." www.rtca.org
2 ARINC Specification 653, "Avionics Application Software Standard Interface," January 1, 1997. www.arinc.com
3 Wind River VxWorks 653 Platform product page. www.windriver.com/products/platforms/safety_critical/
4 Wind River, ACT, and Smiths Aerospace C-130AMP press release. www.windriver.com/news/press/pr.html?ID=296
5 Smiths Aerospace Boeing 7E7 Dreamliner Common Core System. www.windriver.com/customers/customer-success/aerospace-defense/smiths787.html
6 EADS/CASA. www.windriver.com/news/press/pr.html?ID=201
7 John Rushby, DOT/FAA/AR-99/58, "Partitioning in Avionics Architectures: Requirements, Mechanisms and Assurance," March 2000
8 ARINC Specification 653-2, "Avionics Application Software Standard Interface," December 1, 2005. www.arinc.com
9 POSIX Specification, ANSI/IEEE POSIX 1003.1-1995; ISO/IEC standard 9945–1:1996
10 Y. C. (Bob) Yeh, "Design Considerations in Boeing 777 Fly-by-Wire Computers." Third IEEE International High-Assurance Systems Engineering Symposium, 1998. http://doi.ieeecomputersociety.org/10.1109/HASE.1998.731596
11 P. Parkinson & F. Gasperoni, "High Integrity Systems Development for Integrated Modular Avionics Using VxWorks and GNAT." 7th International Conference on Reliable Software Technologies, Ada Europe, 2002. http://link.springer.de/link/service/series/0558/bibs/2361/23610163.htm
12 DO-297, "Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations." www.rtca.org
13 Eclipse Consortium. www.eclipse.org
14 Wind River Workbench product page. www.windriver.com/products/development_suite
15 P. Tingey & P. Parkinson, "Avionics Security." Defense Procurement Analysis, Summer 2003
16 Jean Paul Moreaux, EADS-Airbus, "Evolution of Future Aircraft Data Communications." NASA Workshop on Integrated CNS Technologies, May 2001. http://spacecom.grc.nasa.gov/icnsconf/docs/2001/CNS01_Session_F3-Moreaux.pdf
17 ISO/IEC 15408: 1999, Information Technology—Security Techniques—Evaluation Criteria for IT Security. www.iso.org
18 G. Kuhn, "VxWorks Secure Architecture." Technical paper, Wind River
19 DO-178B, "Software Considerations in Airborne Systems and Equipment Certification." www.rtca.org
20 "Safety Management Requirements for Defense Systems," Parts 1 & 2. Interim Defence Standard 00-56, Issue 3, December 17, 2004, UK Ministry of Defense. www.dstan.mod.uk/
21 FAA Draft Notice, N8110 RSC. www.faa.gov
22 Raytheon WAAS Customer Success Story. www.windriver.com/news/press/pr.html?ID=393

About the Authors

Paul Parkinson is a Senior Systems Architect with Wind River in the UK, where he works with Aerospace & Defense customers. Paul's professional interests include Integrated Modular Avionics and Intelligence Surveillance Target Acquisition and Reconnaissance Systems (ISTAR). Paul blogs on A&D industry issues at http://blogs.windriver.com/parkinson.

Larry Kinnan is a Senior Engineering Specialist for Aerospace & Defense with Wind River in North America. He has responsibility for ARINC 653 solutions and experience with numerous aerospace programs such as the 767 Tanker, Boeing 787, C130-AMP, and other commercial and military aircraft. Prior to joining Wind River, Larry was employed in the medical device design and development community where he was involved in safety-critical device design, development, and deployment. Larry's personal interests range from model rocketry, general aviation, and commercial space flight.

About Wind River

Wind River is the global leader in Device Software Optimization (DSO). Wind River enables companies to develop, run, and manage device software faster, better, at lower cost, and more reliably. Our platforms are preintegrated, fully standardized, enterprise-wide development solutions. They reduce effort, cost, and risk and optimize quality and reliability at all phases of the device software development process, from concept to deployed product.

Founded in 1981, Wind River is headquartered in Alameda, California, with operations worldwide. To learn more, visit www.windriver.com or call 800-872-4977.