PORTING APPLICATIONS TO AN ARINC653 COMPLIANT IMA PLATFORM USING VXWORKS AS AN EXAMPLE

Larry Kinnan, Wind River Systems, Hudson, OH.
Joseph Wlad, Wind River Systems, Alameda, CA.
Patrick Rogers, Ada Core Technologies, Friendswood, TX.

Introduction

With the advent of ARINC 653-1 [1] and the availability of ARINC 653 compliant partitioned systems, most Avionics manufacturers are now faced with migrating or porting their existing legacy applications from a single address space system widely used in Federated Avionics systems to an ARINC-653 system. This also includes combining multiple applications which formerly ran on separate CPU cards onto a single CPU running multiple partitions that emulate multiple virtual CPU's. This consolidation of applications also leads into issues of scheduling and latency effects of the partitions as well as issues of different operating modes (flight, ground maintenance, etc.). This paper will discuss the issues and considerations needed when moving these applications to the new system using Wind River's VxWorks RTOS as the example environment.

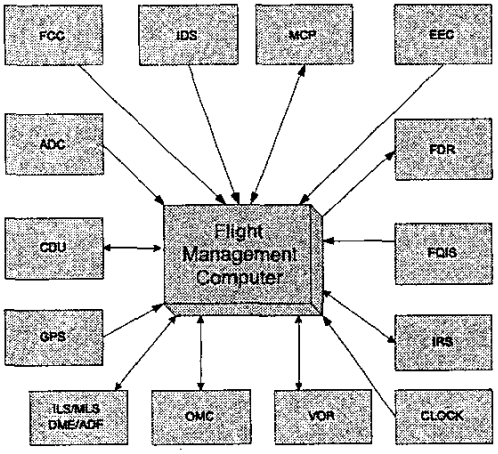

Figure 1. Typical Federated Platform

The Legacy Environment

Figure 1 shows a typical Federated System consisting of a number of subsystems each comprised of their own CPU and housing interconnected via some type of bus such as ARINC-629, ARINC-429 or MIL-STD-1553. Each LRU (Line Replaceable Unit) can be running its own OS (or RTOS) that is then communicating with other subsystems or computer. Typically these were single address space RTOS and applications built into a single module that is then stored in a non-volatile memory area. Each subsystem could be executing on a different processor as specified by the vendor supplying the subsystem. While this assists in localizing faults to a specific LRU, it can lead to a large spares pool as well a presenting numerous integration and interoperation issues.

In the case of a specific LRU using VxWorks, typically the application is linked with the RTOS and then installed as a monolithic object module on the subsystem. The VxWorks single address space RTOS makes use of the fact that kernel and application both run in Supervisor mode and kernel calls are simple function calls by the application. This generally means that the entire module is then certified to a specific level as defined by DO-178B.

The Partitioned Environment

In an ARINC-653 compliant partitioned environment, each partition becomes a virtual LRU or CPU. Each partition is then assigned a time window in which to execute its application. Most modern CPU's have an MMU to enforce this partitioning and this allows multiple applications to be present on a single CPU without affecting any of the other applications in the event of a failure. Taken further, it allows multiple levels of criticality to reside on the same said processor. An example of this environment is shown in Figure 2.

0-7803-8539-X/04/$20.00 © 2004 IEEE

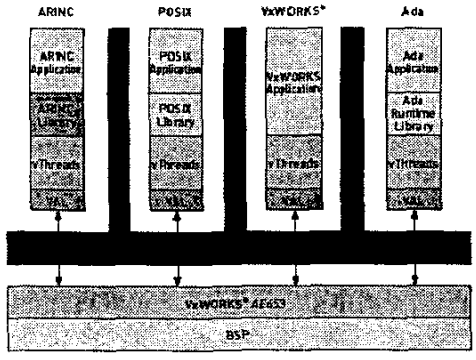

Figure 2. A Partitioned System (AE653)

As can be seen from Figure 2, a partitioned environment can offer many advantages to a developer including the use of separate API subsets and languages in a safe and secure environment. It can also present a developer with a number of challenges when trying to migrate an existing application to this environment.

Migration of an Existing Application

Initial Discussion

To discuss the specifics of migrating an application to a partitioned environment, a certain number of assumptions are made. First, the existing application was written to run on Wind River's VxWorks version 5.x RTOS. The specific release level of VxWorks is not a major concern as will be explained shortly. The assumed migration to a fully ARINC-653 compliant system in this case is also a Wind River product - Platform Safety Critical: ARINC653, which uses the VxWorks AE653 RTOS. This environment provides a significant number of advantages to the developer. The major advantage is that the structure of the AE653 system provides a Partition Operating System (POS) that is based on VxWorks version 5.5 except that it is running in User Mode rather than Supervisor Mode as discussed earlier. For Partition Scheduling and Spatial Partitioning, a Module Operating System (MOS) is used. It also provides the framework for supplying and executing the Health Management (Health Management is a framework that provides monitoring, reporting and remedial action to errors and conditions internal or external to partitions). The MOS allows for partition level restarts (both cold and warm) and management of the configuration records that drive the system initialization (the ARINC653 specification [1] describes the use of configuration record(s) to drive the allocation and initialization of the system and resources). Having a separate POS and MOS provides an environment that allows the developer to ignore the fact that they are running in a partitioned, time sliced environment initially so the migration will only deal with the application and OS interface that the developer is already familiar with, that being the vxWorks 5.x environment. The use of the POS also provides a significant advantage in performance when context changes occur between tasks (threads) running within a partition since no system call overhead to the MOS is required in order for the switch to be accomplished. This is a significant performance improvement over other more monolithic type implementations of a partitioned operating environment that require the MOS handle these type of context switches as well as the partition scheduling.

Migration – Source or Binary

Ideally it would be desirable to just move the existing binary application module in its entirety to the new partitioned environment. This is not feasible for a number of reasons. First of all, if the application is a fully linked module with the OS, this will not be compatible with the AE653 POS since the partition supplies its own OS environment. Secondly, if the application exists as a separate object module, it must conform to the proper OMF (Object Module Format). If the application is built to execute on the same CPU architecture, say PPC604, then the OMF is the same for AE653 (ELF). Unfortunately the move to the more advanced environment necessitated using a different debug format for AE653 (AE653 employs the DWARF2 debug format where versions prior to vxWorks 5.5 used STABS, except for specific architectures). This would mean the module could be compiled to operate in the partitioned environment but would be unable to be debugged, certainly an undesirable situation. Finally, even though that standalone application was built with vxWorks and vxWorks is also the POS, the developer will need to verify that the vxWorks' APIs being used by the application are included in the POS library. To do this verification,

Wind River supplies a concise list of supplied libraries for the POS. This list is provided in the VxWorks AE653 Technology Guide [3] in Section 5.8.

In addition, some API's may not be available for use in the Partition Space. If the application is doing Physical IO to a device, additional considerations need to be made. These issues are discussed in the following section, as are the method(s) for communication between partitions. This will most likely involve the use of ARINC ports rather than say a socket communication or a VxWorks message queue over a bus (network). An Integrator will likely be migrating multiple, separate applications to a single CPU with multiple partitions so this new type of communications will need to be accounted for especially for an application that will be certified at some point. While it may be possible to do this without changing the method of communications, it will only defer the migration work not eliminate it once the developer needs to go to a fully ARINC653 compliant system with or without certification.

If the application is to be certified, one must also be aware that the certified subset of the vxWorks API will be even more restrictive. Also the ARINC653 specification [1] specifically disallows allocation of memory after initialization. While typically not an issue for an application that has already been certified for a flight worthy system, it can be an issue for applications that may have been written for a non-certified environment. Such applications are not unusual with ARINC-653 because the use of an integrated environment is relatively new. Existing non-certified avionics applications provide a number if insights.

Highly portable code was not typically a requirement for existing avionics applications. These applications frequently depend on unique hardware characteristics and facilities. Such dependencies are unavoidable. Similarly, avionics applications may depend on specific implementation details of the compiler and underlying system software. These dependencies were introduced either because programmers were not sufficiently aware of the dependencies or because they had more pressing requirements than portability, or both. In some cases, however, dependencies on software implementation details could have been mitigated or avoided.

Software implementation dependencies affect the cost of reusing the application because a different compiler and underlying system software may be used when moving the application to the newer ARINC-653 environment. This will be the case, for example, with existing applications written in Ada. Many avionics applications are written in Ada because it was designed to support such applications. However, these applications are written in the older version of Ada, known informally as Ada 83. The ARINC-653 Ada compiler supports the new version of Ada, known informally as "Ada 95". Ada 95 is extremely upward compatible with Ada 83. Portable Ada 83 code is very likely both portable and compatible Ada 95 code. However, existing Ada 83 applications may not have been written portably for the reasons described above.

The revision of the Ada language did introduce changes that affect application code, particularly when the original code is not portable. For example, Ada 95 strictly defines the semantics of the sizes of objects and types. These semantics were not rigorously defined in Ada 83 and so portable use was difficult among Ada 83 compilers. Any applications using those semantics in a non-portable way are inherently non-portable with Ada 95 compilers. These dependencies introduce potentially costly changes to the source.

Another issue affecting the cost of certification is that existing non-certified applications may have used features that are expensive to certify. There is an economic trade-off to be made between changing the existing code to remove such features versus retaining those features and certifying their run-time library implementations. Removal of these features can represent a significant expense. For example, existing applications may make extensive use of exceptions, may allocate and deallocate memory during execution, and may have source code that generates implicit loops and implicit conditionals in the object code (such that the source code does not exactly "match" the object code). The compiler and underlying system software for the ARINC-653 environment must support the trade-off process by allowing the system designer to decide whether to change the

code or retain the feature usage, ideally on a feature-by-feature basis.

To migrate applications built with another OS, one would need to map the current OS API's to the corresponding vxWorks APIs via a translation layer or replace the existing OS API calls with vxWorks APIs in the application itself. While not as foreboding as it sounds, it will represent an additional amount of work for the developer up front in the migration effort. Almost by default one would certainly need to work with source code for this situation or link the foreign binary with a translation layer module.

Issues of IO in a Partitioned Environment

Once the initial API issues have been dealt with the developer will be presented with issues of accomplishing IO in the ARINC653 environment. The first issue that needs to be examined is the difference from a Federated system which each application/OS module has control of the IO for their respective subsystem. In the partitioned environment, the developer now faces integration of multiple applications using the same physical IO device(s). Typically this is dealt with by providing an IO Partition that manages IO from the various applications and receives or delivers IO traffic to the underlying hardware. In addition, in a typical ARINC653 environment, interrupts from IO devices are discouraged since it leads to non-deterministic behavior in the system. This is typically mitigated by hardware design that isolates the device bus from the bus that the CPU uses for processing. This will invariably lead to a system in which IO is accomplished by polling usually through a system of buffers or mailboxes that the IO partition containing a manager monitors for incoming IO traffic and also serves as the destination for outgoing IO traffic as well. The second issue is the IO partition also will monitor traffic coming from the partitions as well as delivering the IO data to the respective partitions through their respective ARINC ports.

For data being received from or sent to partitions (inter-partition), the preferred method is to use the ARINC port mechanism (either queuing or sampling depending on the needs of the application). The advantage of this is that even if certification is currently not a requirement, it may be required in the future. By using ARINC compliant ports and other services, the integrator can benefit from the portability and reusability provided by the ARINC 653 specification as well as make use of the certification evidence supplied by the vendor of the OS. Based on the direction the global aviation market is taking, certification will likely be a requirement in the near future, The ARINC ports also provide a well specified interface for all applications to program to rather than using varied an potentially disparate methods and also allows for reuse of portions of the IO code across multiple applications.

Figure 3 shows a simplified view of the ARINC port mechanism. The port mechanism is conceptually the same for either sampling or queuing ports. It also demonstrates how IO traffic goes between partitions as well as in the case of AE653 the IO traffic can also be accessed by the Module OS layer in the event the developer would like to make use of existing drivers that are in the Module OS such as the Ethernet drivers and network stack supplied with AE653 but are not present in the certified version of the OS. It allows for a phased approach to altering the IO model for the applications while maintaining portions of the certified environment in the event certification becomes a requirement later in the project life cycle. The developer only needs to be aware that code that is added to the Module OS will need to be certified to the same level as the highest certification level of the system since it can directly interact with the low level portions of the system. This would be the responsibility of the developer adding the code.

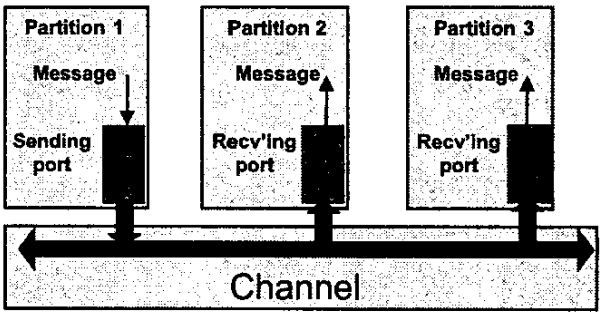

Figure 3. ARINC Port Mechanism

While the IO model appears strange to most real time developers owing to the fact the IO is polled, it really simplifies the design and timing characteristics of the system. The primary issue faced by developers is insuring that the IO partition is scheduled with sufficient regularity to service the IO needs of the other partitions and devices. This usually requires that the IO partition be scheduled multiple times during the ARINC major frame. Also with the ARINC model, the port sizes and depths must be determined and allocated at system initialization time. This is accomplished in the AE653 system by use of the Configuration Record. An overall system design specification is a must to determine the needs of each partition and the IO requirements for the module in general.

The other option for communication is by use of a shared data region that still allows for a certifiable configuration allowing for inter-partition communication. By using these shared data regions it is possible for partitions to share data in a safe manner. It does place responsibility upon the developer to implement a mutual exclusion method to prevent accidental overwrite of data by one partition while being read by another. This is mostly mitigated by the fact that only one partition is active at a given time but consideration must be given to the situation when one partition has a task that starts a read operation and hits the end of its time period before completing this operation. It is then possible for the other partition that also uses the shared data region to overwrite the data thus causing an inconsistency in the data once the original partition is active again. The mutual exclusion issue for shared data access to physical device registers, such as mailbox registers for data transfers, can also be mitigated by the design of the hardware itself.

IO Alternatives using VxWorks AE653

While the ARINC ports mechanism is the preferred method, the amount of work required to convert existing applications to use ARINC ports may not practical during the initial stages of a project. Alternatives exist that allow for easy migration of existing applications by using features of the AE653 OS that are not intended to be certified in the product but exist in the non-certified API set. It is possible to use the Pipe device interface to communicate from a partition to the Module OS by creating a named pipe device in the module OS and then opening that device in the Partition. This makes use of the Global IO device mechanism supplied with AE653 for the development environment. It is also the mechanism provided for doing character IO to the console port device. The advantage is that the named pipe service in VxWorks allows for the use of standard open, close, read and write API calls [2]. A service that the developer provides resides in the Module OS can direct the IO traffic to either an ARINC port, another pipe, a message queue or even output it to the network stack via a socket interface. This allows the full flexibility of the VxWorks IO subsystem while still maintaining the partitioning environment both spatially and temporally. The drawback is that this design makes use of non-certified API calls. This is a tradeoff best left to the developers of the Module rather than forcing them into one specific model of IO. A block diagram of such a service for the network stack using the socket library is shown in Figure 4.

In order to insure that a partition does not become blocked when doing this type of IO operations (IO that relies on the Module OS to perform the IO rather than from within the context of a partition), AE653 provides an optional component named VAL_WORKER_TASKS. These task(s) are native Module OS tasks that execute within the time frame of the partition that they are associated with. They allow the potentially blocking IO operations to executed without blocking the entire partition and then send the results of the operation back to the partition asynchronously. This is discussed in detail in the Platform for Safety Critical Technology Guide [3] in section 4.8 and can be configured by the developer as to the number of tasks required in order to service the IO needs of the partition.

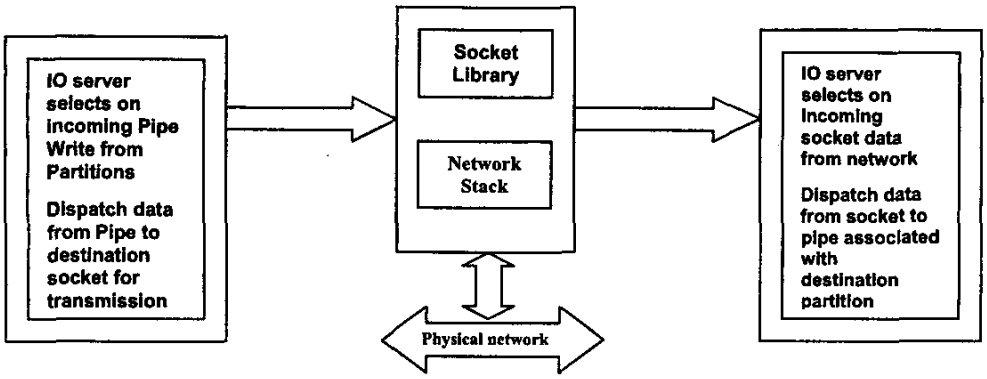

Figure 4. IO Server Block Diagram

Partition Scheduling Considerations

ARINC 653 time partitioning guarantees that each user partition gets a slice of time for execution as determined by the integrator as illustrated in Figure 5. This is usually determined as a worst-case execution time for all the threads within the partition. This can mean that in some cases, that there is CPU slack time within the partition time frame. Threads within a partition will execute on a priority-preemptive basis so RMA can still be applied within the partition. The ARINC 653 time partitioning design works best for systems that have mostly periodic processes (e.g. processing sensor inputs at a 20 Hz rate). Some systems may not be well suited for time slicing, as events may be more asynchronous. In these case, hardware assisted offload may be required in order to satisfy the certification requirements or one may use priority preemptive scheduling across partitions. While from a high level this may seem the simplest approach, it can make certification much more difficult as it may require that all applications be certified to the same level thereby eliminating the benefits of an IMA design.

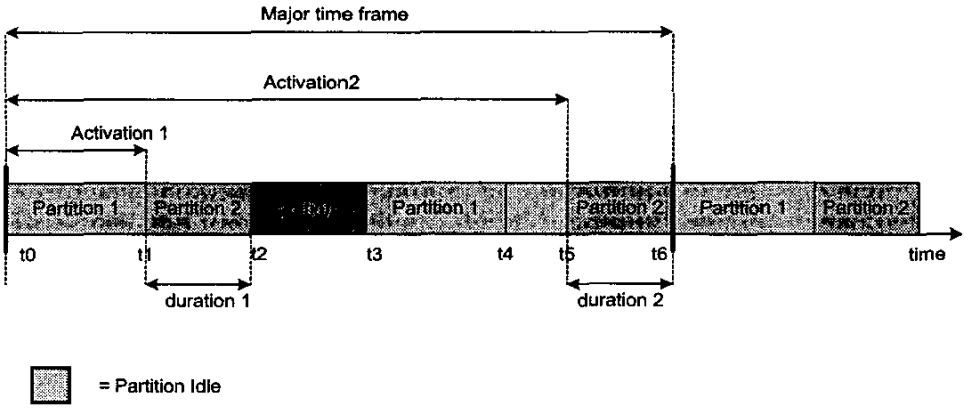

Figure 5. ARINC Partition Schedule Example

Figure 6 provides and example of how one can use the benefits of an ARINC653 compliant system to overcome possible limitations in strict RMA-type designs. The figure depicts 3 applications that run at various rates and durations. For the purpose of argument, one could view these partitions as tasks or as individual partitions composed of multiple tasks. Using the conventional Liu-Layland theorem [4], all CPU utilization should be less than 78% to ensure that all deadlines will be met. However, upon inspection one can see that the actual CPU utilization with these three tasks is 85%, so one can not guarantee that using conventional Priority-preemptive scheduling that all deadlines will be met. One can however, use an ARINC 653 scheduler to guarantee that all threads meet their deadlines.

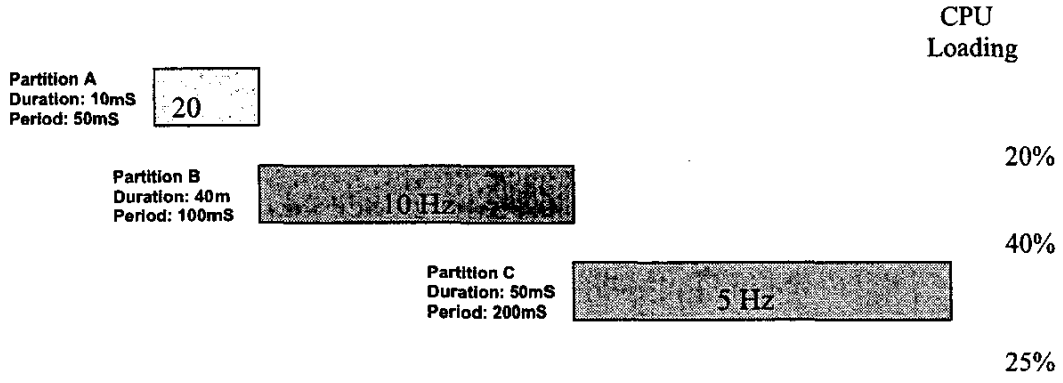

Figure 6. 3 Partition Scheduling Showing 85% CPU Utilization to Accommodate the Scheduled Tasks/Partitions

First, one must determine what the least common multiple of the periods is. In this example, the longest period is 200 milliseconds, thus it is the least common multiple. One could attempt to set up a major frame of 200 milliseconds and a minor frame of 10 milliseconds however, when attempting to heuristically layout the partition schedules using this minor/major frame combination deadlines will soon be missed. This is because the long duration, short rate tasks (Partition C here) tend to impinge on the completion of other partitions. Therefore it is necessary to split all tasks into shorter durations such that the tasks all complete on time.

To adequately schedule these partitions, it is convenient to use the 10 milliseconds minor frame above (Figure 7) as the major frame and to then cycle partitions within this major frame. If one can assign a higher tick rate to the partition, each partition can then be allocated a required number of ticks based on the throughput it needs. In our example here, partition A needs 20% of the CPU, Partition B needs 40% and Partition C needs 25%. One then prorates these based on the minor frame and tick rate. Here, we use 4 ticks for A, 8 ticks for B and 5 ticks for C, which add up to 17 out of 20 ticks needed for these partitions. If we have a 10 ms minor frame, we need to break this minor frame up into 20 ticks, which means the required tick rate will be 0.5 milliseconds, or 200 Hz. The benefit is that it is very easy to enter the scheduling data into the configuration records of the system. It also allows for some scheduling slack in order to handle asynchronous events. The drawback is that a relatively high tick rate may be required to handle long duration, high rate tasks.

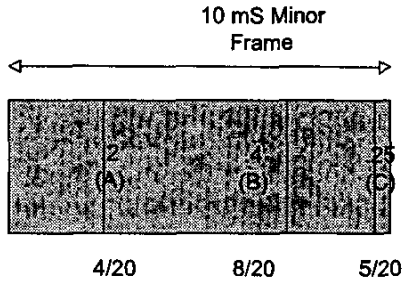

Figure 7. 3 Partition Example using RMA

The AE653 design also allows for use of the standard vxWorks 5.x scheduling mechanisms in order to facilitate legacy application migration within a partition. This includes the normal priority preemptive available in vxWorks as well as the round robin scheduling to be optionally enabled for added flexibility within each partition. This round robin scheduling permits equal CPU time-sharing for threads at the same priority level within the partition. This allows legacy applications that have designed using RMA to be placed into an equivalent operating environment.

Summary

This paper seeks to identify the unique challenges a developer will face in migrating existing applications into the ARINC 653 compliant environment. While some issues are quite obvious, such as module format, source language and existing OS implementations, some are quite specific to the ARINC 653 environment such as input/output and scheduling concerns for both the existing tasks (processes) as well as the newly introduced partition scheduling mechanisms provided by ARINC 653. The developer's challenge is to effectively address these issues in a reliable, cost effective way.

References

[1] Airlines Electronic Engineering Committee, October 2003, ARINC Specification 653-1, Aeronautical Radio, Inc., Annapolis, MD.

[2] Wind River Systems, Inc., June 2004, vThreads API Guide, Wind River Systems, Alameda, CA.

[3] Wind River Systems, Inc., June 2004, Platform Safety Critical: ARINC653 Technology Guide, Wind River Systems, Alameda, CA., pp141-142.

[4] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. 1973, Journal of the ACM

Email Address

Larry Kinnan, larry.kinnan@windriver.com