

MAXIMIZING WAVEFORM PORTABILITY IN A RADIO ARCHITECTURE THROUGH A COMMON HARDWARE ABSTRACTION LAYER MODEL

In previous columns, I have discussed the trade-offs associated with choosing both the signal processing devices that are used in a radio platform and the architectural model for supporting those devices. In this column I thought I would expand on this theme and explore elements of the software operating environment associated with the radio platform that are necessary to support the overall radio requirements. More specifically, I will explore the requirements on the radio's software infrastructure necessary to support the portability of waveform code from radio to radio. A waveform application, for these purposes, will be defined as an assembly of software and firmware components that are deployed on the radio hardware to implement the entire set of radio functions, from the user input to the RF output and vice versa [1]. Components, in this context, encapsulate some functionality supporting the waveform, with well defined interfaces or ports into and out of that functionality [2]. Examples of such components can include ActiveX controls, JavaBeans, common object request broker architecture (CORBA) components, field programmable gate array (FPGA) IP cores, and ExpressDSP™ algorithms [3, 4]. In this context component artifacts include the deployable software and firmware (including HDL code as appropriate) necessary to realize the component's functionality on the target radio platform.

Why a focus on the portability of waveform code? Fundamentally, it comes down to a desire by many companies to maximize the return on investment in their software and firmware development. If waveform code is portable, it allows reuse of that code in multiple different products, thereby lowering overall development cost for those products and achieving a faster time to market or time to deployment. It also reduces the support costs associated with the deployed code by allowing a common code base to be maintained across multiple products. The mechanisms supporting portable waveform code also ease the insertion of new features into existing radio products, thus extending revenues beyond the original product life cycle.

Let me start the discussion on software infrastructure support for waveform code portability with what in some markets is a fairly controversial statement: hardware technology is generally not sufficiently advanced to support the creation of truly portable waveforms. Now that I have irritated the military radio community, let me explain. The signal processing technologies and platform architectures utilized in radio systems tend to vary greatly from program to program, vendor to vendor, and generation to generation. Although efforts can be made to minimize the cost of porting a waveform's software and firmware artifacts, in whole or part, from platform to platform, these differences in hardware architecture necessitate some level of porting effort. The question therefore is "What software infrastructure is necessary on the radio platform to minimize the porting effort?" This question is being explored by a number of programs including the End-to-End Reconfigurability (E2R) program in Europe, NASA's Space Telecommunications Radio Service, and the U.S. Department of Defense's Joint Tactical Radio System (JTRS) [5-7]. The question has also been explored by a number commercial companies, including Texas Instruments in the creation of their ExpressDSP reference frameworks for their digital signal processing (DSP) devices [8]. These efforts show that, in general, improving waveform portability requires the following elements in the software infrastructure:

- A well defined operating environment for each class of signal processing device allowing code to move from

"like processor to like processor" (FPGA to FPGA or DSP to DSP)

- A standard application framework that is used across platforms to set up, tear down, connect, and control waveform components across the various processing devices
- A standard set of application programming interfaces (APIs) for accessing common radio services
- A well defined mechanism for connecting components to support the overall waveform functionality

For general-purpose processors with lots of memory, many of these elements are readily available through constructs such as Microsoft® .Net Framework and the Object Management Group™ CORBA component model [9, 10]. However, the specialized processing and resource constrained devices predominant in many radios, including FPGAs and DSPs, often do not have comparable constructs. Instead, a hardware abstraction layer is generally established within the platform infrastructure to provide APIs for loading and unloading components onto the specialized devices, and providing a mechanism for component-to-component communications.

Of course, as with any technology in this industry, there really is no one size fits all view on hardware abstraction. For example, radio technology developers seem to have four different models for a hardware abstraction layer as it relates to component-to-component communications, each with its own sets of trade-offs in waveform portability, platform performance, and hardware costs. These models are summarized below.

The "No Hardware Abstraction Layer" Model: The use of no hardware abstraction layer is the traditional model employed by embedded developers in the creation of real-time systems. This is often the preferred model because it allows developers to manually optimize how the various software and firmware components within the waveform interact with each other and with the signal processing hardware. Following this traditional paradigm, real-time communications between components are supported at a fairly low level, effectively at layer 1 or 2 of the International Standards Organizations open systems interconnection (OSI) seven-layer model, with the components directly accessing the hardware infrastructure to move data [11].

Components in this model are often fairly monolithic, with a single component encapsulating all of the functional or algorithmic elements instantiated on a given signal processing device. Data formatting between functional elements is facilitated within the component and optimized as appropriate for operation of the waveform. These aspects of the "no hardware abstraction layer" model generally tie component implementation fairly closely to the hardware architecture. As such, although the "worker" code within a component that implements the actual functional algorithms may be portable between like devices, the waveforms themselves generally require a fairly large porting effort because the interfaces between components and component execution are hardware-specific.

The "Transport" Model: One level of abstraction up is a model where the waveform software and firmware components are interconnected through data transports that employ standard APIs to abstract the underlying communications. This type of hardware abstraction operates just above layer 5 in the OSI model, and essentially follows a Berkeley sockets paradigm for moving data between waveform components. In this transport model of hardware abstraction, the various com-

(Continued on page 30)

(Continued from page 28)

ponents within the waveform open logical connections through predefined transports to endpoints on the other components with which they wish to exchange data, and then read and write to those connections as appropriate for waveform operation. The transport interfaces used by components in this model appear the same regardless of the physical transport architecture, whether that transport is a physical “processing device to processing device” interconnect, such as a PCI bus, or a memory mapped logical connection to another component collocated on the same processing device.

In this model the hardware infrastructure is responsible for setting up the end-to-end connectivity through the transports as directed by the components, however a component in this “transport model” must have some a priori knowledge of the remote component that it needs to communicate with in order to establish these connections and to facilitate any data format translation necessary to access the transport interface and move data between components. This makes the component implementation fairly waveform specific. However, unlike the no hardware abstraction layer model, the interfaces to the transports are “standardized,” not hardware-specific, thus greatly reducing the overall porting effort in moving the waveform, as a whole, from platform to platform. In addition, the person porting the waveform has the ability to statically select the specific transport used for a given component-to-component connection, allowing the waveform “porter” to optimize the connections between components for real-time performance. This allows the waveform porter to retain some of the advantages of the no hardware abstraction layer model while significantly improving overall waveform portability.

The “Container” Model: Portability at the component level can be achieved by abstracting not only the interfaces to the transports but also the connections between components. In this model there is no a priori knowledge within a component of the other components to which it is connected or the transports through which it connects, making the component itself portable from waveform to waveform. The component is completely isolated, or contained, by the operating environment of the device on which it is instantiated, with component interfaces operating at layer 6 or higher in the OSI model. The hardware infrastructure in the container model is entirely responsible for component-to-component communications. Data formatting in this model also occurs within the platform infrastructure, not the component. Since support for all permutations and combinations of data types is onerous for resource-constrained processing devices, the number of data formats supported by the hardware will be limited to some fixed set.

The container model is, effectively, an object-oriented software model as opposed to a resource-constrained embedded systems model, designed to minimize the porting effort in moving components from waveform to waveform. This model implies an increase in resources in the hardware infrastructure to support the routing of data packets between components that may be deployed on random devices. Maintaining real-time performance through this infrastructure may require some measure of quality of service on the communication paths, further increasing the resources required in the container model. These increases in required resources may not be appropriate in many size-, weight-, and power-limited systems, and the level of overhead associated with accessing these resources may impede performance sufficiently as to not be appropriate for applications with very tight timing specifications.

Container Standardization: The container model presented above allows for the possibility that each type of processing

Hardware abstraction model	Component connection	Data transport	Platform resources	Porting effort
No hardware abstraction	Component	Component	Low	High
Transport model	Component	Platform	Medium	Medium
Container model	Platform	Platform	High	Medium

■ **Table 1.** Hardware abstraction models for component-to-component communications.

device may support a unique container architecture. The container model paradigm can be taken one step further, however, by standardizing the container implementation across processing devices. This model would use a “standard” container architecture, such as CORBA, on all devices regardless of device architecture, with a “standard” communications protocol, such as the OMG’s General Inter Orb Protocol (GIOP), providing component connectivity [12]. This model has not as yet been widely adopted in deployed radio systems, and may not be mature enough for use in any real radio product.

Table 1 summarizes the hardware abstraction layer component-to-component communications models discussed above, indicating:

- Whether the responsibility for data transport lies with the waveform component or the radio platform
- Whether the responsibility for establishing connections between components lies in the waveform component or the radio platform

It also gives a relative evaluation for each model in terms of platform overhead and porting effort. So which of these models is the most appropriate for a given radio design? Well, again, as with most technologies used in the creation of radio systems, the answer is “it depends.” The trade-offs associated with each model must be weighed in the context of the overall program requirements to decide which model is the most suitable.

REFERENCES

- [1] SDR Forum, “Design Process and Tools Working Group Request for Information,” <http://www.sdrforum.org/public/sdrf-dpt-rfi-final.doc>
- [2] WebCam Components, “About Component-Based Development,” <http://webcabcomponents.com/componentization.shtml>
- [3] Sparx Systems, “The Component Model,” http://www.sparxsystems.com.au/resources/tutorial/component_model.html
- [4] Texas Instruments, “ExpressDSP Algorithm Standard,” <http://focus.ti.com/dsp/docs/dspsupportatn.tsp?sectionId=3&tabId=430&familyId=44&toolTypeId=39>
- [5] End to End Reconfigurability Program, “Work Packages,” <http://phase2.e2r.motlabs.com/workpackages>
- [6] NASA, “Space Telecommunications Radio System Open Architecture Description,” http://procurement.nasa.gov/eps/eps_data/118663-OTHER-001-001.pdf
- [7] JTRS, “Software Communications Architecture Specification,” http://jtrs.army.mil/documents/sca_documents/V2.2/SCA_Release_2.2.pdf
- [8] Texas Instruments, “ExpressDSP Reference Frameworks Software,” <http://focus.ti.com/lit/ml/sprt232d/sprt232d.pdf>
- [9] Microsoft Corp., “Overview of the .Net Framework,” <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpvrintroductiontonetframeworksdk.asp>
- [10] OMG, “CORBA Component Model, V3.0,” <http://www.omg.org/technology/documents/formal/components.htm>
- [11] Wikipedia, “OSI Model,” http://en.wikipedia.org/wiki/OSI_model
- [12] OMG, “General Inter-ORB Protocol,” <http://www.omg.org/docs/formal/02-06-51.pdf>