ARCHITECTURE IMPLICATIONS FOR HIGH CAPACITY, SCA-COMPLIANT RADIO SYSTEMS

Vincent J. Kovarik, Jr.
Harris Corporation
Melbourne, FL

ABSTRACT

The Joint Tactical Radio System (JTRS) Software Communications Architecture (SCA) provides an initial baseline for the common configuration, initialization, and coarse-grained management of a set of resources that, when integrated in a cooperative fashion, form a software-defined radio. Policy evolution has expanded the range of application of the SCA to all communications systems up to 55GHz. The architectural impacts these higher capacity radio systems present are significantly different that those addressed by the initial JTRS procurements. This paper presents an overview of architectural issues and tradeoffs associated with developing an SCA-compliant, high capacity radio system. These issues and tradeoffs will be presented in conjunction with solution spaces enabling the development of a system that meets more than the base set of SCA requirements. An overview of a working reference implementation of a radio system that supports up to 300 Mbps, operates in the 15GHz range, and is configurable and re-programmable under SCA control will be presented.

INTRODUCTION

The initial effort of the Joint Tactical Radio System (JTRS) program was targeted towards the military tactical arena. These radio systems operated within a relatively narrow frequency range from 2 MHz to 2 GHz. The foundation of the JTRS effort, the Software Communication Architecture (SCA), however, is frequency and bandwidth agnostic. Consequently, the SCA may be applied to a range of applications, including radios for both military and commercial systems that operate at frequencies and data rates encompassing a much broader range than originally envisioned.

Data throughput and sampling rates stress the capabilities of architectures proposed in early JTRS Cluster programs beyond their limits. Furthermore, the approach for building an SCA compliant radio system for the underlying hardware architecture of a high-capacity radio has a different problem space and solution set due to a significant reliance on DSP and FPGA processing capabilities.

In order to provide a common frame of reference, consider the abstract implementation paths for a software-based radio.

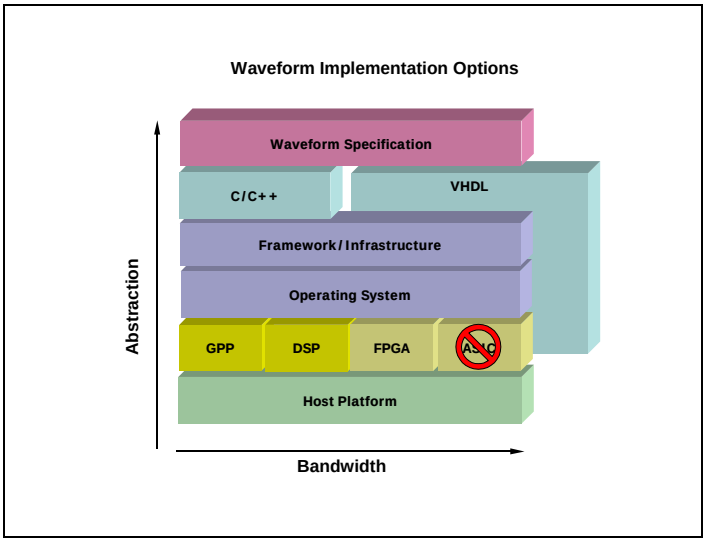

Figure 1. Software-defined radio conceptual framework

Logically, the realization of a software-defined radio can be visualized as shown in Figure 1. The waveform to be realized is specified and, typically, modeled at a high level of abstraction. Then, depending on the demands of the waveform, typically one of two implementation paths is taken towards a processing platform. If the demands of the waveform application are capable of being realized on a General Purpose Processor (GPP) or, if more specialized signal processing is required, a Digital Signal Processor (DSP), then the waveform implementation can be realized using a high-level programming language such as C or C++.

If the throughput requirements of the waveform exceed the capabilities of a GPP and DSP then the waveform can be realized using a Hardware Description Language (HDL) for Very High Speed Integrated Circuits (VHSIC) or VHDL. Through VHDL a digital program can be realized at its most elemental levels as a sequence of instructions forming a state machine. These state machines are then targeted for implementation using a Field Programmable Gate Array (FPGA) or an Application Specific Integrated Circuit (ASIC). As noted in the figure, ASICs are typically not considered as an acceptable target for a software-defined radio due to the fact that they are not re-programmable – a key requirement in a software defined radio. However, the insertion of ASICs is feasible in an

SDR if the underlying communications infrastructure provides certain capabilities.

If we expand upon the middle three layers of the Figure 1, some of the internal components can be identified and these can be organized into four aspects or viewpoints that comprise a software-defined radio. These components, organized by aspect are illustrated in Figure 2.

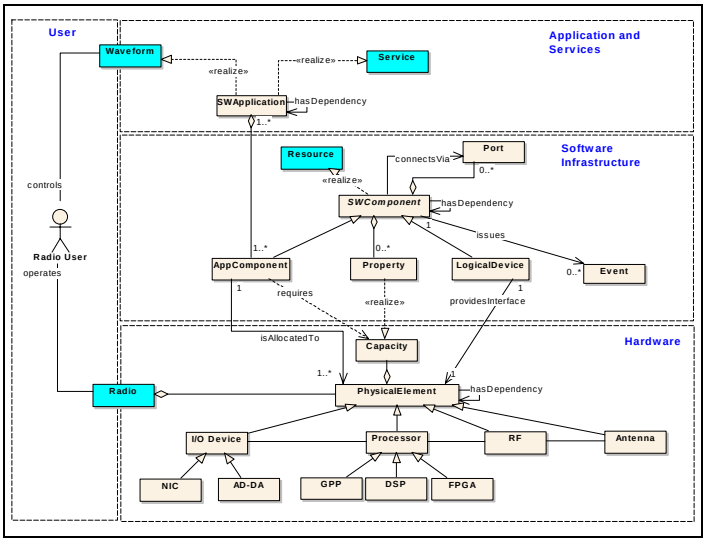

Figure 2. Software-defined radio aspect view

At the lowest level exists the physical hardware. The logical elements of the underlying hardware must be represented in such a fashion that the physical implementation can be managed in a common fashion as a set of physical resources.

From the Software Infrastructure view, the underlying physical components and software components are implemented as a set of software modules providing a means to perform logical operations on the physical devices, such as load a waveform, and between software components, such as establish a signal processing chain.

Then, layered above the Software Infrastructure, is the set of applications and services formed by the collection of software components allocated to the physical devices within the radio system.

Finally from the user's perspective, she either performs operations on the physical radio, e.g. power up or select a waveform, or on the waveform, e.g. select frequency, adjust gain, etc.

The SCA embodies the set of requirements that form the specification of a common software infrastructure for radio systems. What this implies and levies on the radio system developer is discussed in the following section.

SCA COMPLIANCE

As noted in the introduction, the SCA is frequency and bandwidth agnostic[1]. Thus, it is not the target waveform, its operating frequency, or bandwidth that determines whether or not a radio system is SCA-compliant or not. Compliance is determined by the implementation of the SCA specification resulting in a set of common software infrastructure components that, collectively, is referred to as the Core Framework (CF).

The SCA specification, currently at version 2.2.1, defines the interfaces and behavior of this infrastructure. The Core Framework is the realization of the specification. And, when a Core Framework is integrated with a radio system, the system becomes SCA compliant[2].

This concept is illustrated in Figure 3. In both cases, the Core Framework provides the underlying software infrastructure that enables both the tactical radio and the Satcom radio to be SCA-compliant.

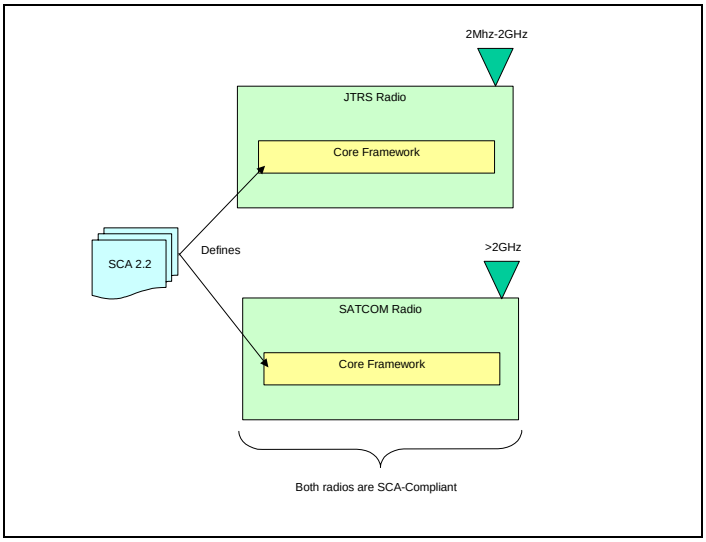

Figure 3. SCA compliance

So, the fundamental test of whether a radio system is SCA-compliant or not can be condensed down to a system that incorporates a Core Framework as the common infrastructure layer within the radio system and the Core Framework enables,

1. The physical devices to be managed, i.e. configured and controlled, through the set logical device interfaces specified by the SCA, and

[1] In fact, it can be stated that the SCA is *application* agnostic as well.
[2] Note that SCA-*compliant* does not imply a system is SCA-*certified*. The term SCA-compliant is used to identify a system that adheres to the SCA specification while SCA-certified refers to a system that has passed the formal certification process defined by the JTRS Technical Lab (JTeL).

2. The applications, i.e. waveforms, to be installed, configured, and controlled, using the set of SCA application interfaces.

It should be noted that there are a significant number of capabilities, behaviors, and requirements that must be satisfied in order to be SCA-compliant, let alone SCA-certified.

Taking the layered diagram of Figure 2 and casting it into a slightly different form, we can visualize the abstraction layers of an SCA-compliant radio. These layers are illustrated in Figure 4.

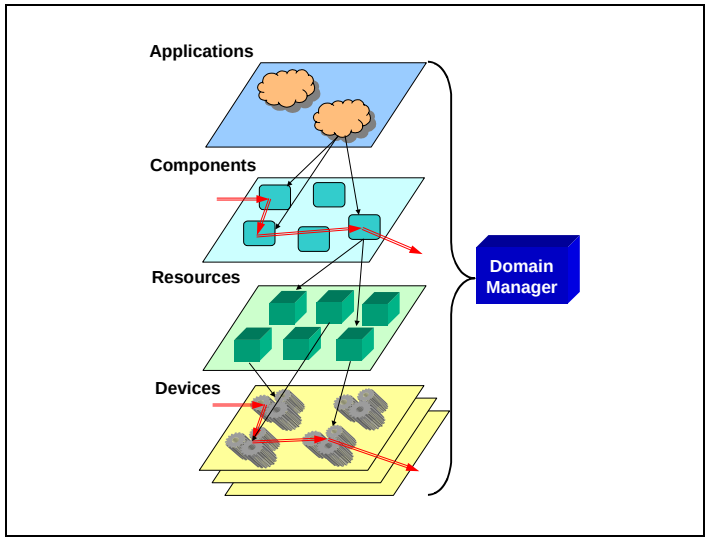

Figure 4. SCA abstraction layers

As noted previously, the radio system consists of a collection of hardware components. This is represented as a collection of devices within an SCA-compliant system and is shown as the lowest abstraction layer in Figure 4. The red arrows denote a signal processing path through the hardware components, i.e. the physical realization of the signal processing path within the radio system.

The SCA employs the concept of resources to capture the processing capabilities and capacities of the radio system. The resource forms a logical abstraction of the physical device. Thus, through the logical device interface, a software module, the higher-level software entities within the radio system can request, allocation of resources to perform specific functions required to realize a particular waveform application.

Above the resource layer is a software component layer. The component layer consists of a collection of discrete, logical software components that form the building blocks of a waveform application. That is, the component concept is used to specify the processing functions that implement a waveform in an abstract manner that is independent of the underlying hardware. These components are then logically connected, as denoted by the directed red lines at the component level in Figure 4, to specify the signal processing path. This logical path is then mapped through to resources and, ultimately, physical devices to realize the waveform application on the underlying hardware platform.

Finally, at the topmost layer, the waveform application is implemented. The application, as described in the previous paragraph, is specified as a collection of components, the connections or path, between the components, and the set of resources required to support those components.

The Core Framework takes this high-level specification of a waveform, represented in eXtensible Markup Language (XML), and performs the logic necessary to find and allocate the required resources, load and configure the resources, establish the connections between the components, and instantiate the waveform. The overall management and control of an SCA-compliant radio is performed by the Domain Manager.

There are several components that comprise a Core Framework implementation. An overview of these components and supporting tools is shown in Figure 5.

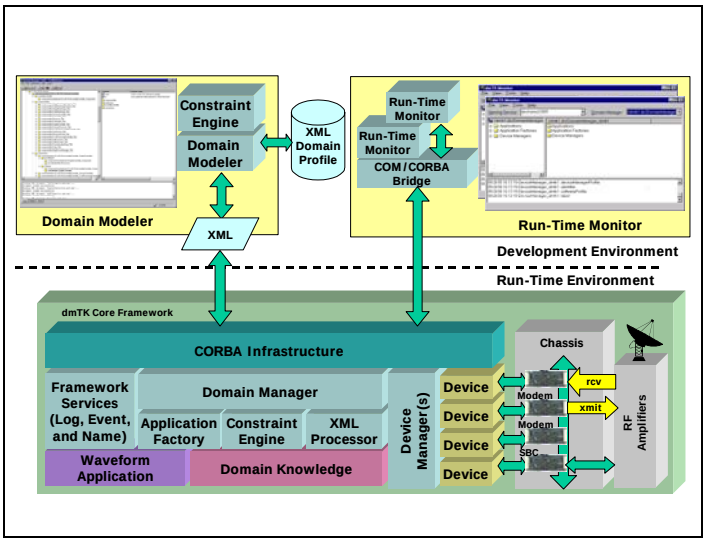

Figure 5. Core framework components and tools

In addition to the Domain Manager, a Core Framework implementation incorporates common services, e.g. file system, logging, event notification, a component called the Application Factory which performs the resource allocation and waveform instantiation, a collection of logical devices providing the control interface to the physical hardware, one or more Device Managers which provide a common reference point for device and services. Finally, a software bus between the components is provided using the Common Object Request Broker Architecture (CORBA) standard.

PROCESSOR IMPLICATIONS

So, within the context of the SCA, how can a high-level waveform specification be realized within an SCA radio? Again, as illustrated in Figure 1, the waveform application may be realized using a GPP, DSP, FPGA or a combination of these processors and, within the context and an SCA-compliant system, the installation, instantiation, and control of the waveforms should be consistent regardless of the underlying processing platform, and, finally, the actual implementation, high-level language or VHDL, should be selected by the Core Framework based on the resources available at the time the waveform is instantiated.

For implementations using a GPP, the ability to install, configure, and connect components to instantiate a waveform is straightforward. Essentially, the waveform is simply another application running as a task within the operating system of the GPP. Connecting each of the components that form the application within a GPP is similarly straightforward as they are simply connected via the CORBA software bus.

However, once the processing demands of the waveform requires that the relative comfort and consistency of a GPP be abandoned for the capabilities of a DSP or FPGA, then the field becomes more complex.

First of all, the interfaces to the DSP and FPGA are different than the GPP. Specifically, the convenient high-level abstraction of the GPP provided by the operating system is not available[3]. Secondly, the mechanism for configuring and controlling a waveform implementation within an FPGA is fundamentally different than under a GPP. In the case of the GPP implementation, the waveform component is a task within the GPP operating system and, as such, can be configured and controlled via function calls.

A VHDL implementation within an FPGA, however, presents a more complex problem. The FPGA does not have an operating system to provide a common abstraction for interfaces. Furthermore, control of the state machine within the FPGA is performed by reading or writing to registers defined in the VHDL. These registers are typically mapped to memory locations within the collection of hardware that forms the radio system.

Finally, level of flexibility of interconnections between the processing components in an FPGA-based system is usually more limited and dependent on the vendor. The data path is governed by the physical topology of the hardware and controlled either through routing logic within the FPGA or using a flex-fabric.

So, the fundamental question becomes how can waveform implementations requiring the processing capabilities of a DSP and, more specifically, an FPGA be integrated within an SCA Core Framework such that the actual implementation hardware is transparent to the configuration and control aspect of the system.

This was the core issue to be solved in order to enable a high-capacity programmable modem developed at Harris to be integrated within an SCA radio system. In the next section, the approaches that were considered are identified, the development path selected is presented, and the potential long-term implications for the SCA specification are discussed.

THE PROGRAMMABLE MODEM

Harris has been conducting internal research and development for the past several years in the development of a programmable modem. Use of the word modem can be a bit misleading, however. While the initial and primary focus was use of the board as a modem, it is essentially a high-capacity digital signal processing board. The high-level architecture of the modem is shown in Figure 6.

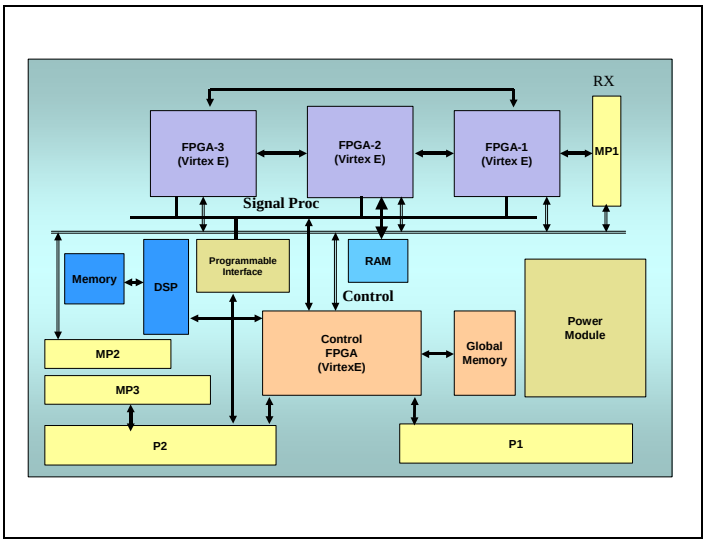

Figure 6. Programmable modem high-level architecture

As illustrated, the core processing capability is furnished by three signal processing FPGAs and a DSP. Each of the FPGAs are interconnected enabling direct communication between any two FPGAs or logic within an FPGA to route data through it to another FPGA or board component.

The board form factor allows it to be incorporated into either a VME or cPCI bus system. Additional processing capabilities, baseband interfaces, intermediate frequency (IF) conversion, and other functions can be added to the

[3] While there are operating systems for DSPs that provide higher-level services, they are not typically as robust as a GPP operating system and, given the FPGA focus of this paper, are not considered.

board through the use of a mezzanine card that plugs directly onto the main board.

In addition to the processing capabilities, the base card provides on-board storage through the Flash memory. This allows several FPGA loads to be stored directly on the board enabling a standard power-up load as well as alternate loads.

Further complicating the issue was the fact that the definition and location of the registers within a VHDL waveform implementation were, essentially, at the whim of the VHDL developer. This implied that the names, locations, and usage of the registers can and will change from waveform to waveform.

So, the core question to be addressed in developing an SCA device interface for the programmable modem was how to achieve the high-level of abstraction defined within the SCA, handle the differing VHDL implementations without placing undue constraints on the VHDL developer, and provide a fine-grained control of the resources that comprise the modem board.

IMPLEMENTING THE SCA DEVICE INTERFACE

Several implementation approaches for the SCA device interface were considered. The initial approach considered was to implement an abstraction library for the modem board. After some consideration this approach was discarded because, for each new waveform, and the new register definitions inherent in the VHDL, the library would be required modification to handle the new definitions. Thus, the library would monotonically expand as new waveforms were incorporated.

While the library could be designed such that the interface were more abstract to minimize the extensions required for a waveform, the library approach still represents essentially a function call or Application Programmer Interface (API) to the board. This would entail extensions to the Device interface that would need to be specified, managed, and linked into the overall application. Although such API extensions are permitted, as noted in the SCA API Supplement, and, in many cases provide a prudent implementation, it was felt that, given the underlying state machine perspective of the FPGA implementation, that an alternative solution would be more appropriate.

The first step was to define the abstract view of the modem board from the SCA perspective. An initial thought was to implement the modem board as a single device. While this approach would have yielded a solution, it was felt that it was not a solution that would provide flexibility as the modem hardware evolved, would provide a fine-grained view and control of the on-board resources, and, finally, was not in the spirit of the SCA.

After some discussion, the approach taken was to represent each discrete processing component on the board as an SCA Device and to implement a Device Manager for the board. This architecture is illustrated in Figure 7.

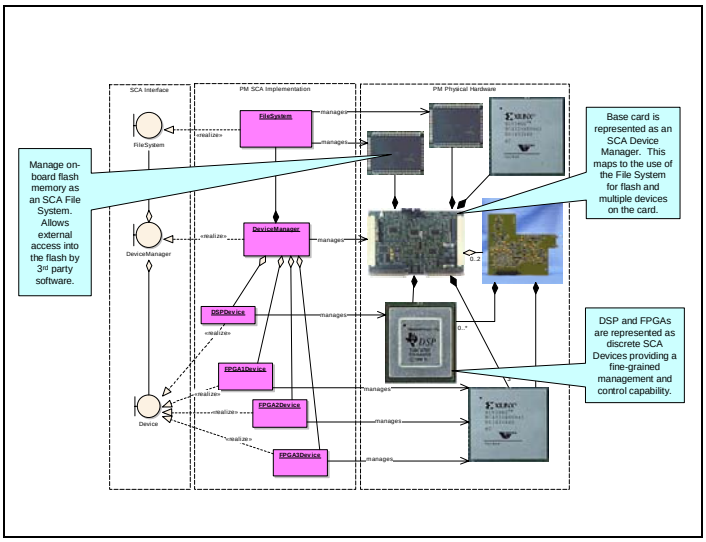

Figure 7. SCA interfaces to the programmable modem

As shown in the figure, each FPGA and DSP has a corresponding SCA Device implementation and a Device Manager implementation for the entire board. As this architectural approach evolved, it became apparent that one of the design options it offered was to incorporate an SCA File System as part of the Device Manager. This solved another problem related to the management of the Flash memory contents. By implementing a pseudo-file system for the Flash memory and providing an SCA File System interface, any SCA-compliant component could now interrogate the Flash memory as if it were a standard file system. This also helped to simplify loading images into the Flash memory.

Although the design of the SCA interfaces for the modem board addressed the basic architectural tenets, there was still the issue of the actual programmatic interface to the VHDL implementation on the FPGAs. Again, the objective was to provide a flexible interface that did not require modifications to a library or API for new waveforms, did not place undue constraints on the VHDL developer, and adhered to the specification and spirit of the SCA.

After considering several alternatives, an approach was developed for configuring and controlling a VHDL waveform implementation in an FPGA that meets the above objectives and, more importantly, has been tested through implementation of an SCA-compliant radio system. This approach is described below.

THE FPGA DEVICE INTERFACE

As implementation approaches were considered, we took a step back and asked what facilities were already available through the SCA that could be applied or adapted with minimal impact to solve the FPGA control problem. After some discussion, the subject of the SCA PropertySet was put forward. The PropertySet allows the definition of arbitrary name/value tuples that can be associated with an entity within the SCA. The property is defined using XML already specified as part of the SCA specification. An example of an SCA property definition for an FPGA register is shown in Figure 8.

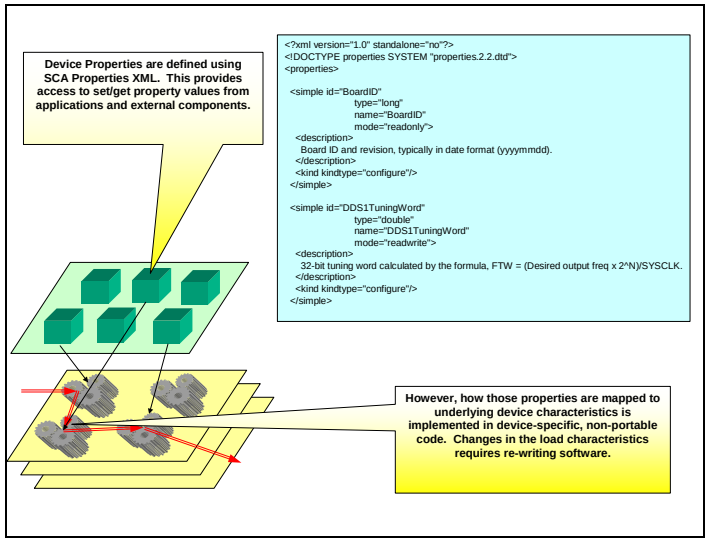

Figure 8. Defining SCA properties

As shown in the figure, the property definition is somewhat limiting in that it allows the specification of a property ID, name, data type, and mode. At first glance it does not appear that properties would be a viable approach. However, the advantage of using properties is that, from an interface perspective, a property definition abstracts and encapsulates the actual property implementation. Thus, from an interface perspective, it presents only the information required to access the property. This was attractive because it offered a mechanism to access FPGA registers in a manner that was consistent with the SCA specification, did not impose any additional interface definitions, and closely matched the concept of an FPGA register as an entity that may be read or written.

So, if SCA properties were to be used, the remaining problem o be solved was how to map the abstract property definitions through to the specific register and memory locations imposed by the VHDL implementation. Again the approach of building an abstraction library for the properties was considered but quickly abandoned.

After several more discussions, the approach chosen was to develop a mechanism for defining the mapping of SCA properties to underlying FPGA registers using a property mapping XML file[4]. This concept is illustrated in Figure 9.

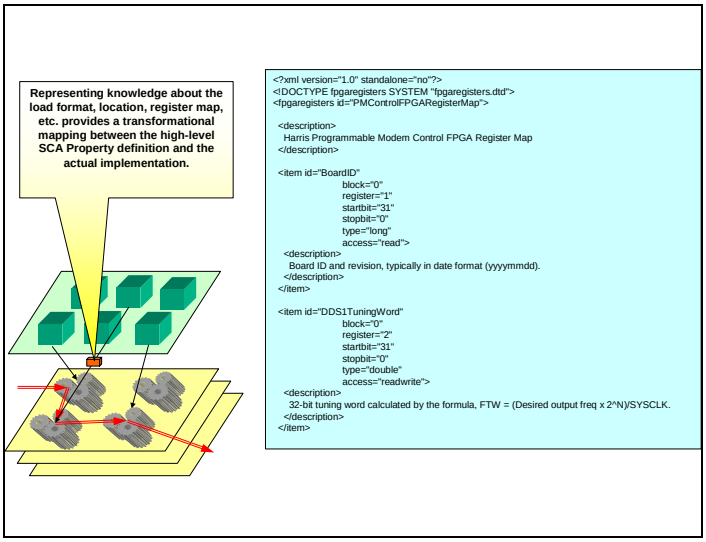

Figure 9. Mapping properties through to registers

As shown in the figure, the property mapping XML file takes the SCA property definition and extends the definition by specifying the information necessary to uniquely identify the register, its location, bit order, offset, etc. By using this simple approach, new waveforms can be brought up under SCA control within days after the VHDL has been completed. All that is required is some time with the waveform developer to identify and define the registers, their use, location, and other characteristics, define the high-level SCA properties, and then develop the mapping XML.

A further benefit is that the underlying device interface code for the modem board does not change for a new waveform. There is no abstraction library to extend or link into existing code. Once the properties and mapping XML files have been developed, along with the standard XML files defining the waveform, the waveform can be installed and instantiated in an SCA radio system. The following section provides a brief example of access to the modem FPGA properties and control of a waveform.

INTEGRATING THE DEVICE

Once the waveform properties have been mapped and the waveform installed, the FPGA properties may be accessed through the standard properties interface. Figure 10 shows

[4] For those with practical experience in applying the SCA, there may be the cry of "not another XML file." The benefits of the approach, however, far outweigh the additional effort required to define the additional XML.

a screen capture of a run-time GUI listing the programmable modem as one of the SCA devices in the system.

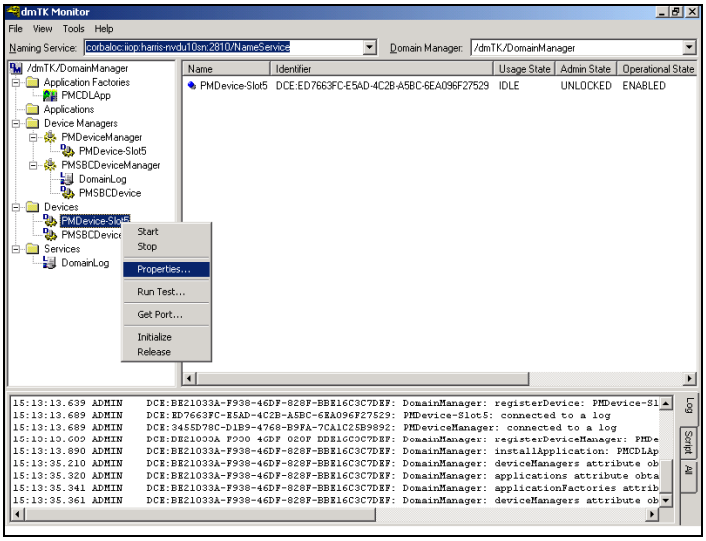

Figure 10. Accessing the modem as an SCA device

Selecting the device and clicking on the Properties… menu selection brings up the properties dialog shown in Figure 11 below.

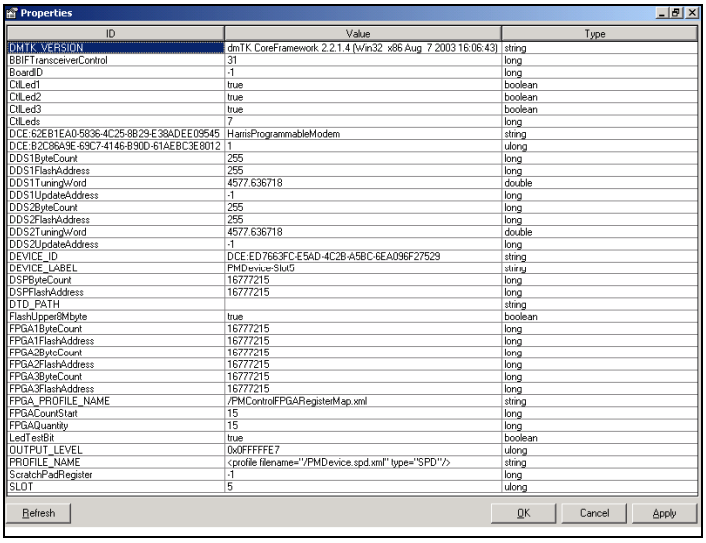

Figure 11. Viewing and modifying modem properties

At this point, direct access to the FPGA registers is available through the properties dialog above. Simply changing a value in the dialog and clicking Apply will write the new values through to the underlying FPGA registers using the configure operation on SCA properties.

Although the above interface shows the ability to define and access properties, it is not a user-oriented interface. As part of our development effort we have successfully implemented and brought up under SCA control both a TCDL and CDL waveform. The TCDL waveform is fully functional and has been demonstrated in conjunction with a legacy TCDL-ELB terminal. This configuration support a symmetrical 10.71Mbps link between the SCA-compliant system and the legacy terminal running live video using IP packets over the link. The CDL is a 274 Mbps implementation.

Instantiation of the waveforms is performed through the Core Framework using the GUI shown below in Figure 12. Switching between the two waveforms entails re-loading the FPGAs and is accomplished in several seconds.

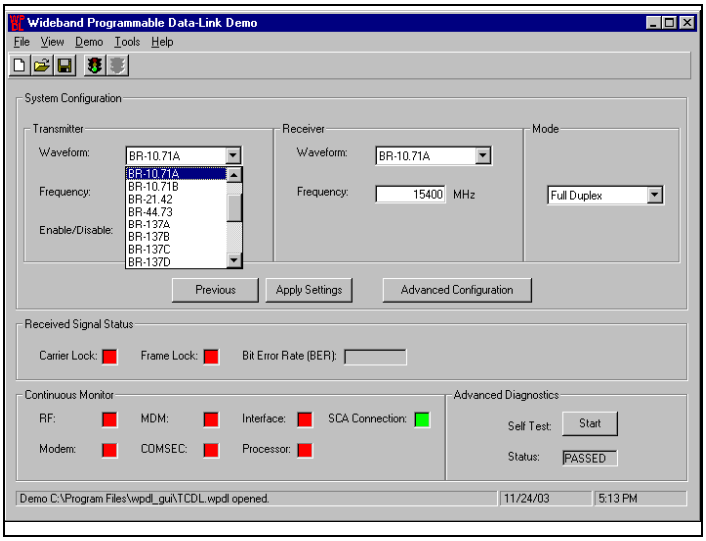

Figure 12. Waveform control interface

SUMMARY

In summary, a flexible approach for integrating FPGA-based waveform implementations has been presented. This approach builds on current SCA specifications and utilizes standard SCA conventions to define specific properties associated with an FPGA.

The approach has been validated through successful implementation and testing of waveforms and interoperability with an existing non-SCA, legacy terminal.

The Specialized Hardware Supplement recently adopted by the JTRS/JPO is an initial step towards a more global solution to the issues presented herein. Work to evolve the initial document needs to continue to be successful and have the long-term benefit envisioned.

Further work is required to address the full range of requirements that would need to be addressed for FPGA-based implementations. However, the approach described above presents a straightforward, flexible, and adaptable approach that maintains the integrity of the SCA while minimizing impacts due to changes in waveform implementation.