ACHIEVING SCA COMPLIANCE FOR FPGA AND DSP COTS HARDWARE

Rodger H. Hosking
(Pentek, Inc.: One Park Way, Upper Saddle River,
New Jersey, 07458, USA, rodger@pentek.com)

ABSTRACT

Developers of JTRS (Joint Tactical Radio System) platforms have been mandated to conform to SCA (Software Communication Architecture) specifications and undergo certification to ensure these requirements have been met.

As a result, many JTRS radio set developers who require FPGA or DSP devices to achieve the necessary performance levels for complex waveforms and multi-channel operation, have been faced with two choices: Either develop custom hardware and build SCA compliance into it from the ground up, or else buy COTS products and try to adapt the standard drivers and libraries for SCA compliance. Both choices pose tangible risk and cost factors.

By addressing the SCA compliance issue during the product development phase as part of the hardware and software engineering effort, COTS (commercial off-the-shelf) vendors can provide SCA-compliant hardware products fully supported with a core framework environment and development tools. This helps reduce risks and costs for the integration effort by JTRS radio developers.

This paper discusses the design considerations for FPGA (field programmable gate array) COTS hardware products, integration strategies of these products into core frameworks, and aspects of development tools for tailoring these products to the specific needs of a radio set.

A simple FPGA-based JTRS transceiver board example is presented for reference.

1. INTRODUCTION

Two essential elements of hardware intended for software defined radio systems targeted for SDR are flexibility and programmability. Hardware products based on GPPs (general purpose processors) not only meet these needs, but many of them support operating systems that are POSIX-compliant. As such, they are capable of operating within a CORBA environment, thus fulfilling the requirements of SCA.

CORBA (common object request broker architecture) assigns processing tasks to a pool of distributed processing resources based on well-defined system management rules and procedures. This effectively decouples the specific signal processing algorithms from hardware structures of the processing engines used to implement them.

Nevertheless, wider signal bandwidths, more complex waveforms (modulation and demodulation schemes), and multi-channel operation are driving hardware designers towards less SCA-friendly devices such as DSPs and FPGAs. While these components can dramatically outperform GPPs with their dedicated and highly-parallel signal processing structures, those same characteristics will not support the necessary POSIX-compliant operating systems.

Putting SCA wrappers around these devices in the form of proxies abstracts the underlying hardware from the rest of the SCA system, allowing FPGAs and DSPs to boost system processing horsepower to be at least somewhat compliant with the SCA environment.

However, these wrappers require layers of software that often compromise the effectiveness of the underlying hardware engines. In fact, this potentially significant trade-off in performance for portability is central to the quest for a standardized method for hardware abstraction.

Towards that end, in August 2004, the SCA Release 3.0 introduced the Specialized Hardware Supplement (SHS) defining SCA connectivity to the HAL (hardware abstraction layer) for DSPs and FPGAs. DSPs were required to implement four operating system calls for control and data transfer functions.

FPGAs required two standardized ports, a sink port for delivering data into the device and a source port for sending data out from the device. In both cases, extensions to the SCA infrastructure are needed to implement these interfaces.

After more than a year, widespread adoption and implementation of the SHS has failed to materialize and additional work on the standard is clearly indicated. Again, one of the major obstacles is trying to come up with a suitably efficient interface between the hardware and SCA.

In order to understand some of the issues surrounding the mission, an FPGA-based COTS product designed for SCA will be presented, followed by a description of how that product is incorporated within a complete SCA development environment.

2. COTS SDR HARDWARE FOR SDR AND SCA

The various hardware elements of an SCA-compliant radio are referred to as *Devices***,** and can include GPPs, A/D and D/A converters, receivers, FPGAs, DSPs, and other types of equipment. From a software point of view, the software proxies that act as interfaces to the hardware are also referred to as devices, since all communication to the underlying hardware must flow through these structures.

Devices are divided into four different classes. Simple elements like an A/D converter that requires basic control, monitoring, and data connections are called *Devices*. A more complex element, like a GPP capable of accepting and executing program code, is called an *ExecutableDevice*.

An FPGA is capable of performing a signal processing function but cannot really execute program code. Instead, the FPGA must first be configured for a particular task and so it is called a *LoadableDevice.* The fourth class of device is a combination of *devices* and so is termed an *AggregateDevice*.

2.1. COTS Hardware Example

One example of a single hardware product capable of meeting the definition of three classes of SCA *devices* is the Pentek Model 7640 Dual Channel Software Radio Transceiver. Shown in Figure 1, the 7640 includes two 105 MHz 14-bit A/D converters, two 500 MHz 16-bit D/A Converters, and a 6-million gate Virtex-II Pro FPGA containing two PowerPC processors.

Other 7640 resources include 512 MB of SDRAM, a four channel digital down-converter and a digital up-converter, both ASICs. As a PCI card, the board plugs directly into a PCI slot of a desktop PC. All communication with the board is performed across the PCI bus to the rest of the system.

Each of the two A/D converter inputs can be connected directly to an RF tuner, which amplifies the antenna signal and down converts the signal frequency to IF (intermediatefrequency), typically below 200 MHz. Each of the two D/A converter outputs can drive the IF inputs of an analog RF up converter and power amplifier suitable for driving a transmit antenna.

Based on the *Device* classes outlined above, the A/D and D/A converters of the Model 7640 can be considered as simple *Devices* and the FPGA as a *LoadableDevice*.

One might think that the two PowerPC processor engines within the FPGA could be candidates for *Executable Devices*. However, it is unlikely that all of the required operating system and CORBA structures are possible within the constraints of this specialized hardware environment. In fact, it is much more reasonable that the PowerPCs might be used as *Loadable Devic*es. In this case, a C program compiled for the PowerPC could be loaded into associated RAM and then executed upon command, analogous to the configuration code loaded into an FPGA.

2.2. Roles of the FPGA

FPGAs are incorporated into many software radio hardware products because they offer several key advantages. First, as a replacement for "glue" logic, their flexible I/O pins can mimic various logic levels to implement a wide range of high-performance interfaces to components, buses, memory devices and communication ports.

In the hardware example shown, note that all of the resources are connected directly to the FPGA. Not only does the FPGA provide the necessary logic interfaces, it also provides flexible and reconfigurable interconnections between the resources.

In addition, the FPGA includes specialized hardware structures including SDRAM controllers for the three

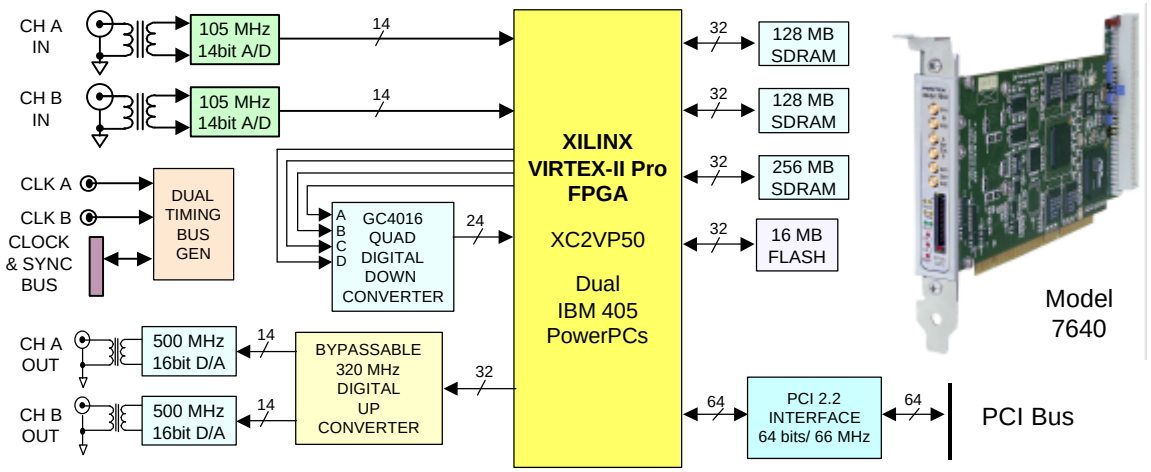

Figure 1. Software Radio Transceiver PCI Board for SCA Development and Deployed Systems

memory banks. These are commercial IP (intellectual property) cores installed in the FPGA as part of the product infrastructure. An advanced dual clocking, timing, and synchronization system built from FPGA logic simplifies data acquisition, time stamping, and supports multi-channel operation.

All of these features are supported with general purpose libraries and drivers so that application programs can setup operational parameters, data flow paths, triggering and timing modes, and data formatting. Up to this point, however, the product could have been implemented with dedicated logic, and still provide all of the same functions. So, in its first role, the FPGA simply makes the hardware design more efficient.

The second role of the FPGA is to provide custom signal processing resources afforded by the wealth of DSP structures including flexible block RAM, hardware multipliers and logic slices. Customers can add their own IP or install commercial off-the-shelf IP cores for specialized tasks, often saving months of development time and offloading signal processing chores from other processors in the system. In this role, the newly-acquired function of the FPGA is still often fixed during deployed operation.

In a third role, however, the FPGA is reconfigured during system initialization or during runtime based on the current needs of the application. Here, we are approaching a feasible facility for SDR.

The fourth role involves exploitation of the on-chip PowerPC processors, that can be equipped with an operating system, and programmed to execute C code. Since they are surrounded by a wealth of device interfaces, memory and real-time signal processing hardware, the PowerPC cores may not be allocated as freely as a generic processor within a pool of GPPs, but those immediate resources may offer an extremely efficient data flow pattern for real-time signal processing tasks.

It is important to realize that a single FPGA is capable of assuming all four of these roles at the same time, but the last two roles are of greatest interest for SDR and SCA.

3. SCA SOFTWARE ENVIRONMENT

How can the example COTS software radio board described above be incorporated within an SCA-compliant environment suitable for development and deployment for JTRS?

Additional objectives for this environment are that it must be low cost, employ industry standard hardware and software components, and be scalable for a wide range of applications. It must include provisions for developing different radios, building new waveforms, creating new applications, and controlling and monitoring radio facilities through a GUI (graphical user interface).

In addressing this mission, the strategy was to approach the problem in a stepwise fashion. The first step was to start with an existing SCA-compliant development platform based on a desktop PC with all of the SCA software structures in place. The platform must also include a reference waveform, a simple I/O device, and a simple application. After a careful survey of several different approaches, the SCARI++ Software Suite from CRC (Communications Research Centre) of Canada was chosen.

3.1. SCARI++ Software Suite Overview

SCARI++ is a full-featured C++ implementation of the SCA Core Framework version 2.2. As a flexible core framework implementation, it is completely configurable, with special emphasis on ease of use. It included several unique features helpful for radio introspection and application debugging.

SCARI++ offers a Component Development Library (CDL) which is used to create application and device components. The CDL implements complex SCA requirements allowing the user to concentrate on the development of applications or devices. For example, the CDL offers a Multithread Safe framework for query/configure services as well as for the device capacity model. The CDL effectively reduces the quantity of code that developers must write and helps meet the SCA requirements.

Completing the SCARI++ Suite is the SDR Development Toolset. Fully integrated with the CDL, it provides the developer with a comprehensive set of tools to design, run, debug and deploy SDR systems.

3.2. SCARI Component Development Library

The CDL has been designed to provide an effective method of rapid application/device creation for SCA developers. The CDL implements several common functionalities that must be part of every SCA component.

CDL automatically handles the SCA *PropertySet* interface through a generic implementation of the Resource interface. The CDL handles the complex validations that must be performed (required by the SCA) when a *Resource* or *Device* is configured or queried. The CDL handles the mandated capacity model (and other state behaviors) through a generic implementation of a *Device*.

Using the CDL, developers can specialize generic properties. The CDL supports the required two kinds of properties (*configure and allocation*) as well as every property format (*simple, simplesequence, struct, and structsequence*).

3.3. SCARI SDR Development Toolset

As part of the SCARI Software Suite, CRC has developed a number of software tools to assist in the development of SDR projects. These tools are closely integrated with the SCARI++ core framework implementations to provide an even better interaction. The Toolset includes five tools to enable application creation through each stage of development.

Component Editor helps create all metadata files for any kind of SCA component. This GUI is form-based, simple to use, and prevents the creation of invalid metadata. The component editor completely abstracts the XML complexity of the domain profile.

Code Generator is a plug-in to the component editor GUI, used to generate most of the SCA IDL Implementation Code normally created manually. This plug-in is used to generate C++ code for Resources and Devices.

Waveform Application Builder (WAB) is a GUI used to construct or modify waveform applications. Using the WAB, applications are created by dragging components from a toolbar onto an application canvas.

Node Boot Builder (NBB) similar to the WAB in look-and-feel, the NBB is used to define a hardware platform configuration in terms of SCA devices and ports. The NBB automatically generates the XML descriptor file to make the platform SCA compliant.

Radio Manager is used to control and configure a SCA radio. This GUI offers a graphical block diagram view of the radio nodes and applications. It displays a hierarchical view of the radio node components.

3.4. SCARII Audio Device and Example Application

The Radio Manager controls the four basic components of an SCA radio: the *DomainManager*, *DeviceManager*, *ApplicationFactory* and *Application*. The *DomainManager* registers all of the components in the radio, creates interconnections between them, installs and controls applications and offers introspection and user interface facilities to the radio.

The *DeviceManager* handles all control and activity of a single Radio Node, which represents the group of devices that together form the operation parts of the radio.

In the example application provided with the system, the Radio Node consists of the Pentium processor as the *ExecutableDevice*, an audio *Device* implemented using the system audio sound port, and a Log Port for monitoring and debugging.

The Application used in the example is the AudioEffect application which accepts audio from the input sound port, adds a controllable echo, and then delivers the output to the output audio port.

4. PORTING COTS HARDWARE TO SCA

SCARI met all of the requirements of the first step in the mission to port the Model 7640 into an SCI environment. Using the SCARI development tools, the second step was to substitute the audio input and audio output ports on the PC motherboard with the A/D and D/A converter ports of the 7640.

The SCARI Component Editor was used to create a new SCA *Device* for the 7640. All of the parameters for controlling the board were imported from the standard C-callable library and drivers supplied with the COTS product. Memory maps for all of the registers and memory resources and low-level routines for manipulating control and status bits were also imported. Templates were defined for testing valid bit combinations and parameter values to prevent entry of out-of-range settings.

Once all items had been entered using the graphical user interface, the Component Editor was generated Component Descriptor XML files for the 7640 *Device*, fully consistent with SCA requirements. These files include the SPD (Software Package Descriptor), SCD (Software Component Descriptor), PRF (Device Property File), and SPD.PRF (SPD Property File).

The Node Boot Builder was used to build a new radio node consisting of the 7640 *Device*, the *ExecutableDevice* (the Pentium processor) and Log Services. It creates the DCD (Node Assembly Descriptor) file, which describes all components within the node.

Once the node is created, the DCD allows all aspects of the radio node to be started and controlled by the *DeviceManager*.

At this point, the node is now ready for deployment by the *DomainManager* which, under control of the *RadioManager*, launches and runs the *DeviceManager*, the *ApplicationFactory* and the *Application*.

4.1. SCA Compliant Development Platform

The 7640 hardware product by itself cannot be considered an SCA-compliant offering, However, by bundling this product with all of the necessary SCA infrastructure the resulting system can be claimed to be SCA-compliant.

Such a system is the Pentek SCA 2510 SCA development platform. Comprised of hardware and software components compatible with the Software Communications Architecture (SCA) standard, the SCA 2510 is a PC-based system running under Linux. It includes an installed (CRC) SCARI++ SCA Core Framework, Component Development Library and SDR Development Tools, plus the Pentek SCA Board Support Package for the Model 7640 for all of the board-specific components.

This complete platform offers a low cost solution suitable for waveform developers, application developers and system integrators. In order to support scalability to larger platforms, additional 7640s can be installed in the PC. In addition, the identical architecture is available in the 7140 PMC module, and the 7240 (6U) and 7340 (3U) CompactPCI modules, all extremely appropriate for large, deployed multi-channel military and commercial systems.

5. NEXT STEPS

Currently underway is the development of a representative waveform component for FM modulation and demodulation. Since the system audio device is still an available component for a radio node as the example provided with the SCARI package, the receiver can deliver demodulated data to the audio output port.

A suitable analog RF front end targeted for the commercial FM broadcast band of 88 to 108 MHz, for example, could be used to drive analog input of the 7640 A/D converter. The 105 MHz A/D converter is easily capable of digitizing this entire 20 MHz bandwidth. Digital down conversion using the ASIC DDC or one implemented within the FPGA tunes and filters the signals of interest, delivering base band signals for demodulation.

Likewise, the SCARI audio input port could be used as the input to an FM modulator component, followed by ASIC or FPGA digital up conversion to the FM broadcast band and then D/A conversion. The relatively low power at the analog D/A output connector of the 7640 could drive an antenna directly for reception by a nearby FM broadcast radio.

Once the next revision of the SCA 3.1 Special Hardware Supplement is released, it will be used to construct a *LoadableDevice* using the FPGA of the 7640.

Future roadmap efforts planned for this product are *LoadableDevice* employing one or both of the PowerPC processor cores inside the FPGA. Once these efforts have been completed, the components will be incorporated as part of the SCA 2510 platform.