

# Extending the SCA Core Framework Inside the Modem Architecture of a Software Defined Radio

LEE PUCKER AND GEOFF HOLT, SPECTRUM SIGNAL PROCESSING, INC.

## Abstract

Extension of the SCA core framework inside of the modem architecture of a software defined radio requires special consideration for managing the non-CORBA-enabled devices. This is especially true of devices that maintain direct hardware connections between each other in support of the high-speed low-latency communications required by many waveforms. This article illustrates the application of the SCA core framework for these types of modem architectures, including aggregating devices in support of direct hardware interconnects between components and the incorporation of a switched fabric communications infrastructure within the overall modem architecture.

## Introduction

A key enabler for software defined radio (SDR) technology is the Software Communications Architecture (SCA) developed by the Modular Software-Programmable Radio Consortium (MSRC) under contract to the Joint Tactical Radio System (JTRS) Program Office [1]. This architecture “objectizes” the radio structure, and defines a standard application framework (referred to as the *core framework*) for instantiating and connecting the waveform objects associated with each radio channel.

The SCA is designed to ensure portability of waveforms across the various radios in the JTRS family. There are limits, however, on the level of portability supported. The SCA does not constrain the modem architecture, allowing the use of any combination of general-purpose processor (GPP), digital signal processor (DSP), and field programmable gate array (FPGA) devices the radio developer deems necessary within the modem to support the physical layer implementation of the target waveforms. As such, one radio developer may choose to implement the modem architecture using a set of processing devices that are connected together in a specific manner. This architecture may be fundamentally incompatible with software components developed by another radio vendor targeting a different set of devices or interconnect topology. Waveform portability in an SCA-compliant radio can therefore only be guaranteed at the modem interfaces, as defined in the SCA 2.2 API supplement [2].

That said, by extending the SCA core framework inside the modem, a standard mechanism is provided for setting up, tearing down, and controlling software components representing the algorithmic elements of a modem application running on the physical devices within the modem architecture. In addition, the core framework enables the connection of these software components in a manner that maintains hardware independence while ensuring that the modem implementation meets all performance parameters. Core framework support within the modem thus extends waveform portability by allowing wave-

form developers to retarget algorithms for a specific modem application across a range of processing devices while maintaining a common hardware-independent application interface. This article describes how the SCA core framework can extend inside the modem architecture in this manner to support improved waveform portability beyond the modem interface.

## Modem Architecture

In general, the use of GPP devices such as PowerPC processors is indicated wherever possible within the modem architecture of an SDR. These types of devices support a POSIX-based operating environment and a programming model that allows for Common Object Request Broker Architecture (CORBA)-based communications interfaces, both of which are required by SCA and serve to maximize the portability of software components targeted to the devices. While it is anticipated that eventually all modem processing may be supported on GPPs, at this time this is not practical in system implementations for two primary reasons:

- GPP devices are available with adequate processing performance for many waveforms. However, the power utilization and heat dissipation of these devices is prohibitive in many size-, weight-, and power-limited systems. As a result, DSPs are typically required to supplement the GPP to meet modem architecture power budgets.

- Processor-based devices such as GPPs and DSPs employ a serial processing architecture that is not appropriate for many front-end preprocessing functions such as channelization (digital down- and upconversion), signal equalization, and radio frequency (RF) tuning. This problem is compounded when dealing with wideband waveforms, since the bandwidth of these waveforms may exceed the usable throughput of the serial processing architecture. The use of an FPGA-based processing element, which gives near application-specific integrated circuit (ASIC) performance in a programmable device, is thus often required in the modem architecture.

SDR architectures addressing these issues are well documented [3–6]. These architectures are summarized in a generic model illustrated in Fig. 1. In this model, an FPGA is used for the high-speed front-end modem preprocessing, with either a GPP or DSP performing the baseband signal processing, depending on the size, weight, and power limitations imposed on the modem subsystem. Analog-to-digital (A/D) and digital-to-analog (D/A) converters may be considered part of the modem or part of the RF transceiver subsystem. When a GPP is incorporated into the modem architecture, waveform link layer processing may be supported directly by the modem; conversely, when a DSP is employed in lieu of a GPP, a common “host” GPP may be provided outside of the modem architecture that is shared across all modem channels.

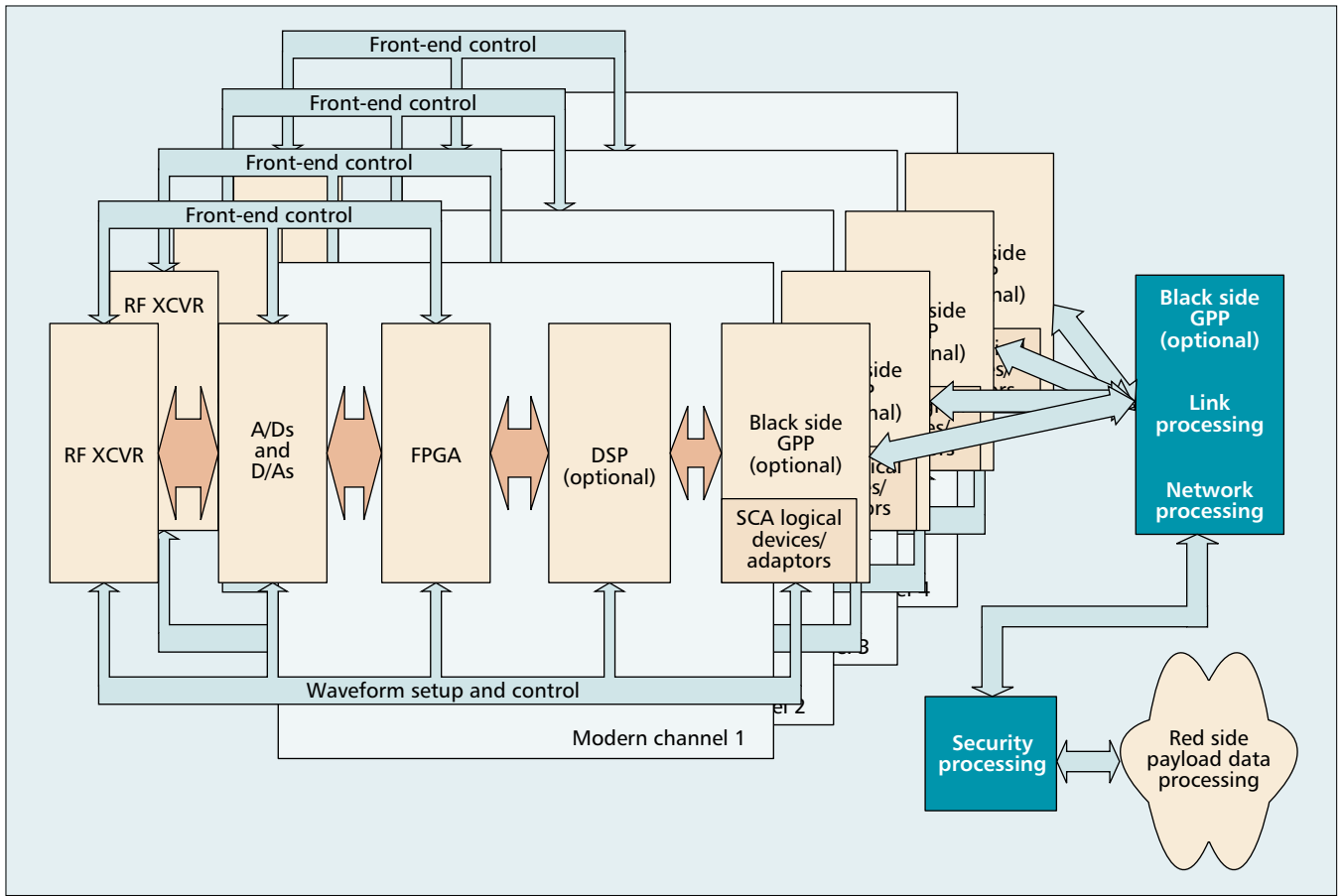


FIGURE 1. Typical radio hardware architecture for a multichannel SDR.

### SCA Support for Processing Devices Inside the Modem

Support for the SCA core framework within this type of architecture requires that each of the processing devices within the radio be exposed to the CORBA-based logical software bus. This bus is used to set up and tear down software components on these devices, and to connect and control these software components through well-defined CORBA Interface Definition Language (IDL) interfaces. Processing devices such as FPGAs and DSPs that do not support CORBA-enabled communications each require an SCA logical device adaptor that resides on a CORBA-enabled processor and acts as a proxy for the non-CORBA-enabled device within the confines of the core framework. These logical devices form a bridge between the SCA logical software bus and hardware-specific application programming interfaces (APIs).

The logical devices required to represent each modem channel within the context of the SCA are:

- A) An executable logical device running on the modem GPP to act as a software proxy for the GPP. This device will be used to execute SCA application resources on the GPP.
- B) An executable logical device running on the modem GPP to act as a software proxy for the FPGA. This device will be used to load (since an executable logical device is also a loadable logical device) an application resource targeted to the FPGA. Such a resource will consist of a firmware core to be loaded onto the FPGA itself and optionally a controller or proxy for the FPGA core to be executed on the GPP.

C) An executable logical device running on the modem GPP to act as a software proxy for the DSP. This device will be used to load an application resource targeted to the DSP. Such a resource will consist of code to be loaded onto the DSP itself and optionally a controller or proxy for the DSP code to be executed on the GPP.

Finally, a device manager is required for the logical devices that will run on the modem GPP. The device manager is instantiated on bootup and will in turn instantiate each logical device. The logical devices then each register themselves with the core framework's domain manager, which manages the overall radio platform.

To illustrate how these logical devices will be used, consider an example where we wish to instantiate the simple SCA waveform application depicted in Fig. 2. This application will run on one of the modem channels identified in Fig. 1, and

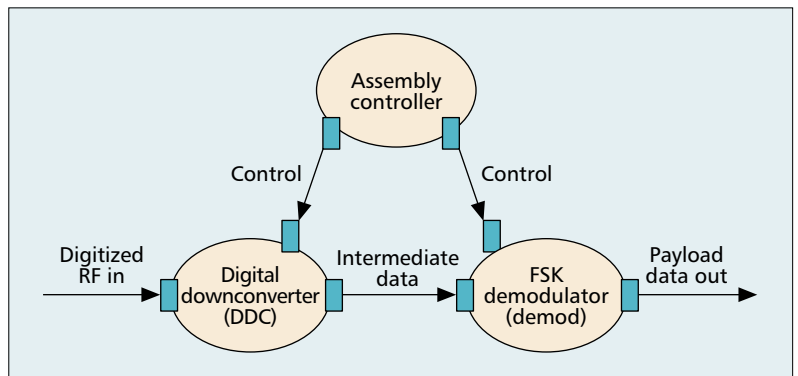


FIGURE 2. Simple receive only modem application.

use an off-the-shelf FPGA digital downconverter (DDC) core and legacy DSP code for frequency shift keying (FSK) demodulation. It will be necessary to wrap these standard components as SCA application resources exporting the required CORBA interfaces. In addition, the SCA requires an assembly controller resource through which the outside world may interface with the waveform application. Obviously, the DDC resource must reside on the FPGA and demod resource on the DSP; however, since the assembly controller communicates exclusively through the CORBA bus, it may reside anywhere.

The data channel between the DDC and demod components will be supported through a direct low-latency connection between the FPGA and DSP devices. This is typical in a modem implementation, since the overhead introduced by CORBA on such connections is often prohibitive, especially for wideband waveforms or waveforms supporting high-rate frequency hopping. Note therefore that the port connection between the DDC and demod application resources is not used for signal path data, which is supported exclusively through the use of local API calls. It is important, however, to maintain the portability of the waveform application by providing dummy port connections between the DDC and demod resources to mimic the operation of these software components on the logical software bus. These port connections may be used to direct the establishment of data paths over the direct connections between the processing devices, or merely as placeholders for future CORBA-enabled links.

To deploy this waveform application on a specific modem channel, the radio's user interface invokes the core framework's application factory's *create* operation. When this occurs, the application factory performs a variety of tasks that instantiate the waveform on available devices within the radio based on the software assembly descriptor (SAD) for the waveform application. As a first step in this process, the application factory locates, for each software component required by the waveform application, an available logical device that

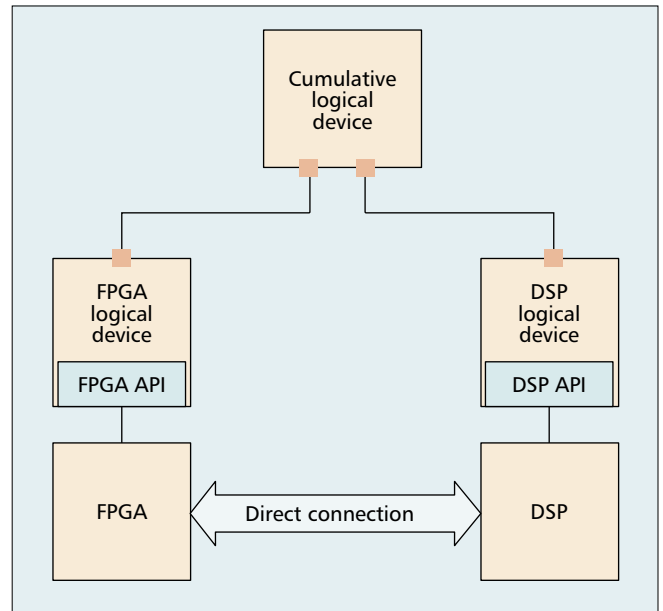


FIGURE 3. Use of a cumulative logical device to aggregate the FPGA and DSP logical devices.

can support that component by means of the *allocateCapacity* operation. The problem inherent in this device selection is that the FPGA and DSP devices identified in this operation may not necessarily have a direct connection between them: the FPGA could be on one modem card, the DSP on another. To address this issue in a generic manner, there are two possibilities:

- Force the selection of specific processing devices by passing the device assignments to the application factory as an argument of the *create* command. This technique is not terribly useful in a multichannel modem, since it requires a sepa-

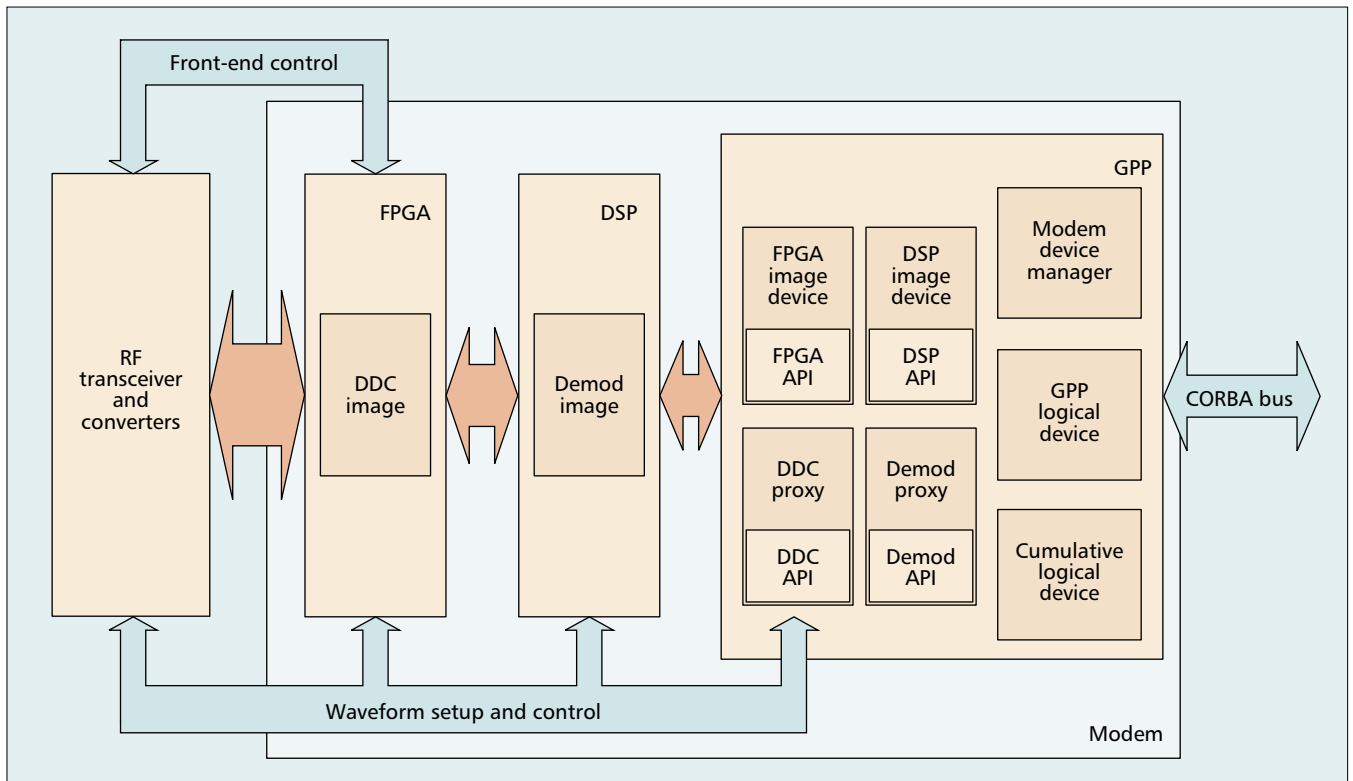


FIGURE 4. Software component deployment for the example waveform application. Core framework components are in deeper green.

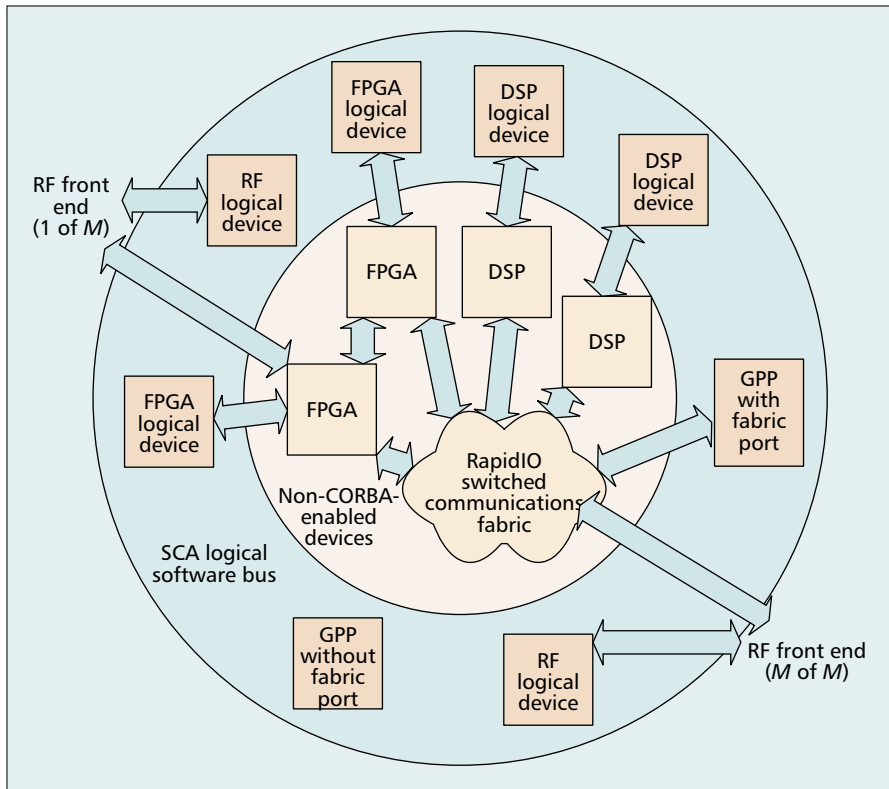


FIGURE 5. Use of RapidIO switched fabric interconnect in an SCA-compliant radio.

rate configuration for each instantiation of a common waveform that is specific to a set of devices.

- Instantiate a logical device that aggregates each valid FPGA and DSP pair into a new logical device. The device manager on the modem GPP will establish port connections between this new cumulative logical device and the local FPGA and DSP logical devices, as appropriate, to allow the cumulative logical device to act as a device facade on the logical software bus (Fig. 3) by delegating its operations to one or both of its subordinate devices. This technique allows the individual DSP and FPGA logical devices, as well as the cumulative logical device, to be exposed to the application factory, which allows a great deal of flexibility in assigning device resources for other waveforms.

Through the use of this cumulative logical device, the tasks that are performed by the application factory to instantiate the modem application include:

A) Allocating capacity on the cumulative logical device for the DDC component by calling *allocateCapacity*. The cumulative logical device delegates this call to its associated FPGA logical device for the DDC image that will be loaded onto the FPGA and for a software proxy for the DDC that will run on the local GPP.

B) Allocating capacity on the cumulative logical device for the demod component by calling *allocateCapacity*. The cumulative logical device delegates this call to its associated DSP logical device for the demod image that will be instantiated on the DSP and for a software proxy for the demod that will run on the local GPP.

C) Executing the DDC component on the cumulative logical device. The cumulative logical device delegates this call to its associated FPGA logical device, which in turn instantiates the software component that will act as the proxy for the DDC image and loads the DDC image on the FPGA.

D) Executing the demod component on the cumulative logical device. The cumulative logical device delegates this call to its associated DSP logical device, which in turn instantiates

the software component that will act as the proxy for the demod, and loads the demod image on the DSP.

E) Calling *getPort* on the DDC proxy to get the “provides” port for communications with the assembly controller

F) Calling *getPort* on the assembly controller to get the “uses” port for communications with the DDC proxy

G) Calling *connectPort* on the assembly controller to connect its “uses” port to the DDC proxy’s “provides” port

Steps E through G assume that instantiation of the assembly controller has occurred on a remote processor prior to the establishment of the port connections. These three steps are repeated for the connections between the demod proxy and the assembly controller. A diagram illustrating the final component deployment within the modem for this waveform is provided in Fig. 4.

The port connections between the assembly controller and the processing resources are used to control the various functions within the DDC and the demod via the logical software bus. For example, the following procedure would be utilized to change the center frequency of the DDC using the estab-

lished ports following a remote call to the assembly controller from the radio’s user interface:

- The assembly controller calls the change center frequency function of the DDC resource, passing the new center frequency in the argument of the call.
- The DDC resource uses a local API call to communicate with the FPGA to change the center frequency of the DDC.
- The FPGA signals the DDC resource through a local API when the change is complete.
- The DDC resource signals the assembly controller that frequency has changed. This could be done as a return on the control call from the assembly controller, or as a separate acknowledge call.

## Switched Fabric Interconnects

The need to support cumulative logical devices in the radio core framework component deployment to provide support for low-latency connections between critical processing elements inherently limits the flexibility of the radio architecture, and may lead to an increase in the overall cost of the radio platform. One answer to this problem that has been discussed in the literature [6, 7] is to incorporate a high-speed switched data fabric, such as serial RapidIO, between the processing elements, as shown in Fig. 5. These interconnects provide a low-latency any-to-any connection between the processing elements, providing the maximum in flexibility while supporting virtually any waveform.

Core framework support for a switched fabric interconnect is provided through the logical devices for each processing element. Each logical device defines one or more connections between the processing device it represents and the switched fabric through which it wishes to communicate. Connections between application software components are then made via the logical devices on which they are loaded. For example, if the simple application in Fig. 2 was to be retargeted to a

switched fabric architecture, it would be connected as shown in Fig. 6.

These connections, which are expressed using the Extensible Markup Language (XML) in the software assembly descriptor (SAD) for the waveform application, are supported through the provision of a fabric port object on the logical device, which exports the CF::port interface and identifies a logical channel into the fabric. On a create command, then, the application factory:

- A) Locates, for each software component to be instantiated, an available logical device that can support the software component by means of the *allocateCapacity* operation. This operation will reserve, among other things, the necessary ports into the fabric to support that component.
- B) Instantiates each software component, and connects it to a fabric port on the logical device on which it was instantiated by means of the SAD port identifier *devicethatloadedthiscomponentref*. Note that since the software component and fabric port will always be collocated, there is very little overhead on the CORBA connection between them.
- C) Connects together the two fabric ports that communicate with one another to establish a connection path through the switched fabric. This mechanism for utilizing the high-speed fabric is fully consistent with the SCA specification and thus allows for SCA-compliant connections between devices using switched fabric technology.

## Conclusion

The approach described above extends the SCA core framework inside the modem. It outlines a standard mechanism for setting up, tearing down, and controlling the software components that could be implemented on a variety of heterogeneous processing devices within the modem architecture. The described methodology is fully SCA-compliant, and supports connecting the software components, as necessary, in a manner that maintains independence of the hardware architecture while ensuring that the modem implementation itself meets all performance parameters. In addition, this methodology maps well to an almost infinite number of variations in modem architecture, allowing the SCA core framework to be utilized as the standard application framework for a variety of commercial, government, and military programs, reducing both the overall costs associated with those programs and the time to deployment for the developed solution.

## References

- [1] SCA 2.2, [http://jtrs.army.mil/pages/sections/technicalinformation/fset\\_technical\\_sca.html](http://jtrs.army.mil/pages/sections/technicalinformation/fset_technical_sca.html)
- [2] SCA 2.2 API supplement, [http://jtrs.army.mil/pages/sections/technicalinformation/fset\\_technical\\_sca.html](http://jtrs.army.mil/pages/sections/technicalinformation/fset_technical_sca.html)

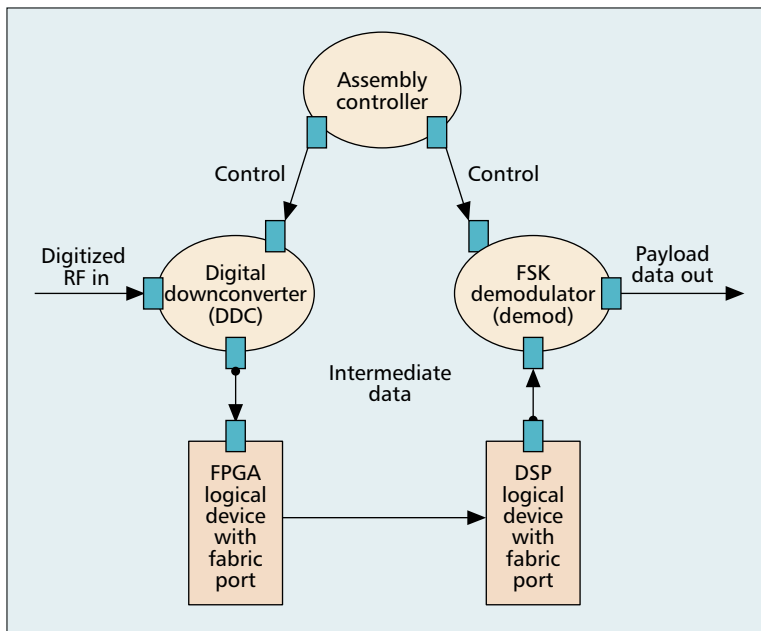


FIGURE 6. A simple application using switched fabric.

- [3] A. Akhurst, "Scalability Aspects of the SRA," SDR Forum contrib., 25 May 2001, [http://www.sdrforum.org/2001\\_docs.html](http://www.sdrforum.org/2001_docs.html)
- [4] D. Murotake, "JTRS SCA Platform Hardware Scalability," SDR Forum Contribution, 13 Nov 2001, [http://www.sdrforum.org/2001\\_docs.html](http://www.sdrforum.org/2001_docs.html)
- [5] SCA Training Course Day 3 — Waveform Design, [http://jtrs.army.mil/pages/sections/technicalinformation/fset\\_technical\\_waveforms.html](http://jtrs.army.mil/pages/sections/technicalinformation/fset_technical_waveforms.html)
- [6] D. Dohse et al., "Successfully Introducing CORBA into the Signal Processing Chain of a Software Defined Radio," *COTS J.*, Jan 2003, pp. 32–37.
- [7] D. Murotake, "Use of Switched Fabrics in Implementation of Software Defined Radio Smart Antenna and Interference Cancellation Signal Processing," *Proc. SDR Forum 2002 Tech. Conf.*, vol. 2, 12 Nov. 2002.

## Additional Reading

- [1] P. G. Cook and W. Bonser, "Architectural Overview of the SPEAKeay System," *IEEE JSAC*, vol. 17, no. 4, Apr. 1999, pp. 650–61.

## Biographies

GEOFF HOLT is a senior software engineer at Spectrum Signal Processing Inc. With 10 years of experience in the field of real-time systems, his areas of expertise and areas of interest include embedded communications infrastructures and middleware for distributed systems. He has Bachelor's and Master's degrees in engineering from the University of the Witwatersrand, South Africa. He has been a member of the Institute of Electrical Engineers, United Kingdom, since 1989.

LEE PUCKER [M] is chief technology officer for Spectrum Signal Processing Inc. He provides integral leadership and technical direction for Spectrum's future products to meet the demanding customer requirements of tomorrow. His expertise and areas of interest include software defined radios, architectures for high-performance digital signal processing, and communication system design. He received his B.Sc. in electrical engineering from the University of Illinois, and his M.Sc. from Johns Hopkins University.