Part 1: Design and implementation of an SCA core framework for a DSP platform

By Carlos R. Aguayo Gonzalez, Francisco M. Portelinha, and Jeffrey H. Reed

(This article is based on a paper presented at the SDR Forum Technical Conference November 2006, Orlando, Florida.)

Editor's note: This is Part 1 of a two-part article. Part 2 will run in the next issue of Military Embedded Systems.

The authors present the design and implementation of the SCA 2.2 Core Framework for a TI DSP platform and provide the rationale behind design decisions and initial profiling results.

The Software Communications Architecture (SCA) was developed by the Joint Tactical Radio System (JTRS) program of the U.S. Department of Defense (DoD) to standardize the development of Software-Defined Radio (SDR) technology. The SCA was developed to enhance system flexibility and interoperability, while reducing development and deployment costs. As with most emerging technologies, early implementations of SCA SDRs have struggled to meet performance, cost, size, and power requirements. Arguably, many of these struggles have their origin in the assumption of a modular, distributed platform based on General Purpose Processors (GPPs) performing all signal processing.

Traditionally, developers have relied on Moore's law to overcome processing power shortcomings. However, in order to deal with the challenges of implementing SDRs, it is necessary to use technology as efficiently as possible. DSPs, for instance, are specialized microprocessors designed specifically for real-time digital signal processing. They have been used for decades in the communications industry to implement radios in resource-constrained environments. In spite of this, DSPs have been relegated as secondary elements in the SCA, requiring a Hardware Abstraction Layer (HAL) for connectivity. Ongoing improvements in development tools and middleware have leveraged DSP technology to fully support the SCA. By following this approach, the flexibility and reusability brought by the SCA are complemented by the cost and power efficiency of DSPs. If taken to a logical extent, this approach could eliminate the need for a GPP on certain SDR implementations.

System architecture

Our system's general software structure is shown in Figure 1, where three different components of the SCA Operational Environment (OE) are shown: Core Framework (CF), CORBA middleware, and Operating System (OS). The last element of the SCA OE, Services (for example, Log, Event, and Naming Services), was not considered for the initial implementation. The Open-Source SCA Implementation Embedded (OSSIE)[1], developed by Wireless@Virginia Tech, Mobile and Portable Radio Research Group (MPRG), was used as the CF.

The ORB used in this project is PrismTech's e*ORB SDR C++ version for DSP[2]. This is an optimized, modular implementation of Minimum CORBA as standardized by the Object Management Group (OMG). e*ORB also supports the Extensible Transport Framework (ETF), which allows custom transport plug-ins.

DSP/BIOS was selected as the OS. It is a scalable real-time multitasking operating system designed specifically for the

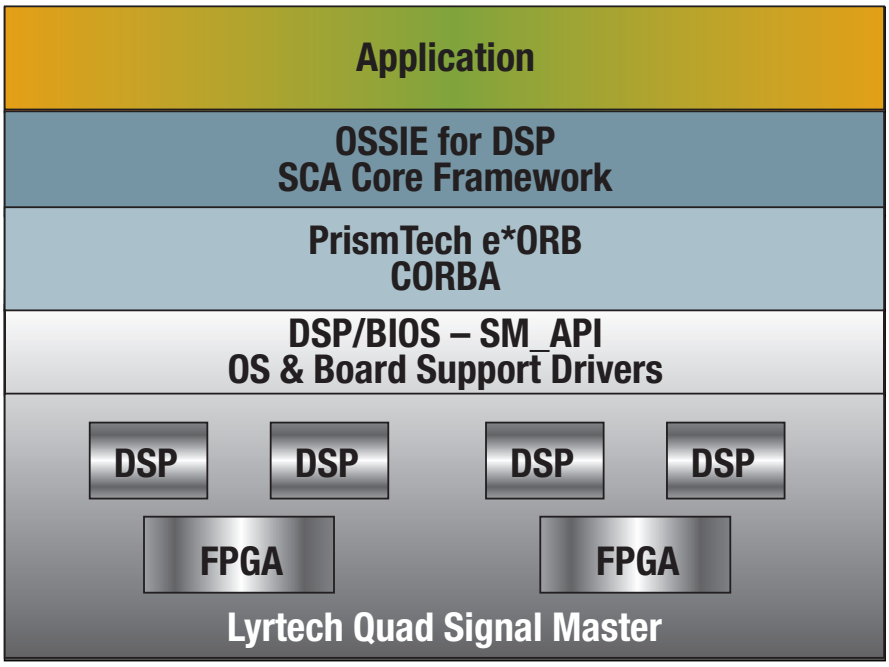


Figure 1

TMS320 family of DSPs[3]. It supports preemptive multithreaded operations because of a real-time scheduler, and provides memory management modules for low overhead, dynamic memory allocation. Note that DSP/BIOS is not POSIX compliant, as required by the SCA.

The target hardware platform for this project was Lyrtech's SignalMaster Quad C6416[4]. This high-performance board contains four TI TMS320C6416T DSPs and two Xilinx Virtex-II FPGAs divided into two clusters (one FPGA and two DSPs each). The system runs at 720 MHz and has 128 MB of SDRAM memory per DSP. There is a high-speed bus between both FPGAs implemented using LVDS. The communication between two DSPs within the same cluster can be implemented using shared memory or FastBus, a proprietary protocol developed by Lyrtech. Only DSPs were used for signal processing and framework functionality. FPGA operation was limited to inter-DSP communications.

OSSIE C64 implementation

The key task for the success of this project was porting the C64 platform of the original version of OSSIE, which runs on an x86 platform running Linux and uses omniORB as middleware. The development platform was Code Composer Studio (CCS), an integrated development environment for TI DSPs, with version 5.1.0 of its Code Generation Tools. This particular version lacks the Standard Template Library (STL) and has limited support for C++ exceptions. In the absence of exception support, we used CORBA Environment variables coupled with a set of macros, distributed as part of e*ORB, for error handling.

A very important aspect to consider is the absence of a Memory Management Unit (MMU) in the C64. The MMU is responsible for handling memory access requests. It takes care of virtual memory management, paging, memory protection, and bus arbitration. Its job is to take pieces of dispersed physical memory and present them to the requesting process as a contiguous block. As a result of using a MMU-less platform, all memory management is the responsibility of the developer. Certain OS calls, such as fork(), are not supported.

Another important development area was porting all scheduling calls to the preemptive, multithreaded DSP/BIOS. The main difference from a traditional fair-share OS is that the active task with the highest priority will be scheduled for execution, no matter how many other tasks are waiting, or for how long. This characteristic allows deterministic execution, crucial in real-time systems, but makes the developer completely responsible for task scheduling and priority assignment.

The SCA specification requires the reading of the XML Domain Profile at runtime to obtain deployment and configuration information (for example, the implementation of ApplicationFactory must read a Software Assembly Descriptor file in order to know which components are included in a given waveform and respective connections). Parsing XML is a complicated task for a DSP, and there are not many tools available to help with it. To facilitate development, reduce memory requirements, and speed execution, a two-step parsing scheme was implemented. In this scheme, shown in Figure 2, an offline translation of the XML files into a simplified format was performed. This two-step parsing did not affect the traditional SCA waveform design cycle and only added one extra step at installation time. The savings in time and complexity, along with the uncompromised portability of the resulting waveforms, justified this decision.

Because our hardware platform did not have long-term storage capability, only a partial implementation of a file system was developed. The host computer's hard drive and file system were used to store the translated domain profile files. Due to limited I/O functionality in the runtime support library, directory-related interfaces (such as mkdir, rmdir, mount, and unmount) were not implemented.

The SCA specifies two equivalent mechanisms to launch software components: ResourceFactory and ExecutableDevice. In our design, the former was used to launch components in the host node (the one that loaded DomainManager), and the latter was reserved for remote nodes and required a DeviceManager. Due to the absence of a transport layer, only ResourceFactory was implemented. This was done using DSP/BIOS scheduler, with a new task being scheduled every time a new component instance was required. This behavior requires that all software tasks are loaded in program memory before they can be scheduled. ResourceFactory was in charge of managing the new task's priority.

Sample application

Framework functionality was demonstrated by deploying two sample applications. These applications were intended for demonstration purposes and nothing else. No extensive signal processing was performed. The main goal for these applications was to verify framework operation and corroborate the feasibility of deploying SCA-compliant waveforms onto the C64 platform.

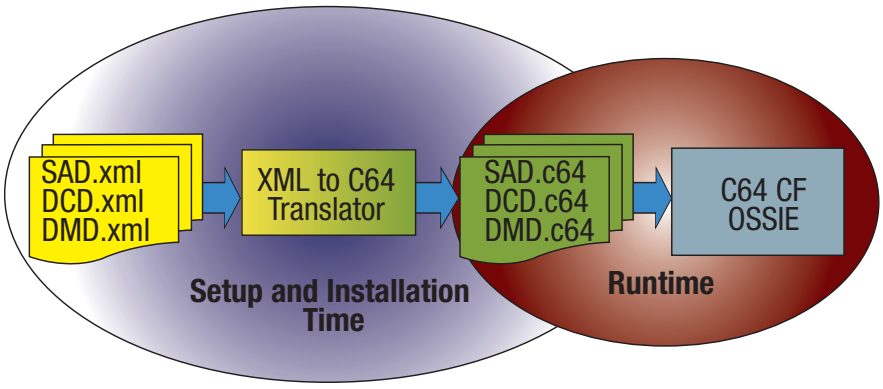


Figure 2

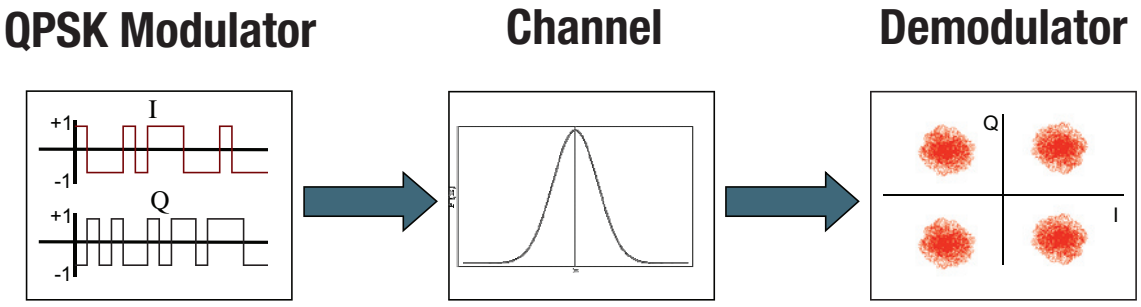


Figure 3

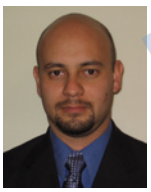
The first application includes three simple components: Binary Phase-Shift Keying (BPSK) modulator, channel, and demodulator. The BPSK modulator generates a random stream of 1s and -1s. The stream is passed to the channel component, which adds Gaussian noise to the in-phase and quadrature components of the stream. The demodulator only displays the constellation diagram of the signal. The second waveform includes a Quadrature Phase-Shift Keying (QPSK) modulator instead of BPSK. Figure 3 shows a graphical representation of the second waveform. Both waveforms were successfully deployed on a single-chip configuration using the ResourceFactory interface to launch the components.

Acknowledgments

This work was supported by Texas Instruments, PrismTech, Mercury Computer Systems, Lyrtech, and the National Institute of Justice.

References

1. Open-Source SCA Implementation Embedded, http://ossie.mprg.org.
2. PrismTech, www.prismtech.com.
3. Texas Instruments Inc., www.ti.com.
4. Lyrtech Signal Processing, www.lyrtech.com.



Carlos R. Aguayo Gonzalez *is a graduate research assistant at the Mobile and Portable Radio Research Group (MPRG) of Virginia Tech. He is a PhD candidate in Electrical Engineering at the same university. He received his BS from Instituto Tecnologico y de Estudios Superiores de Monterrey (ITESM), Mexico, in 2000 and his MSEE from Virginia Tech in 2006. He was employed by Mixbaal in Mexico from 2000 to 2002 as a software design engineer. At MPRG, he has participated in several projects implementing SDR. He is a member of the OSSIE development team. His research interests include SDR implementation, validation, verification, and testing.*

Virginia Tech
Wireless@Virginia Tech
432 Durham Hall
Blacksburg, VA
540-231-2966
caguayog@vt.edu
www.wireless.vt.edu

Dr. Francisco M. Portelinha *is an assistant professor at The Federal University of Itajuba (UNIFEI), Brazil. He received his BSEE from the National Telecommunications Institute (INATEL) of Santa Rita do Sapucai and his MSEE and PhD from the University of Campinas (UNICAMP). Francisco worked for four years as a telecommunications engineer in the Brazilian Bank Automation System. He also worked at the Telecommunications Research and Development Center (CPqD-Telebras) as a telecommunications researcher, where his work included an optical time domain reflectometer and digital radio development. He was a visiting scholar at the Virginia Tech MPRG and is a member of the OSSIE development team. He has more than 20 years of experience in DSP design, systems development, and consulting for Texas Instruments of Brazil, FURNAS, Nansen.*

Universidade Federal de Itajuba
Av. BPS, 1303
37500-176 Itajuba, MG, Brazil
portelinha@uti.psi.br
www.unifei.edu.br

Dr. Jeffrey H. Reed *is the Willis G. Worcester Professor in the Bradley Department of Electrical and Computer Engineering. He served as MPRG director from 2000 to 2002 and serves as director of the newly formed umbrella wireless organization Wireless@Virginia Tech. Jeffrey's areas of expertise include software radios, smart antennas, wireless networks, and communications signal processing. He is the author of two books,* Software Radio: A Modern Approach to Radio Design *(2002) and* An Introduction to Ultra Wideband Communication Systems *(2005). He received the College of Engineering Award for Excellence in Research in 2001 and received an award in 2004 from the SDR Forum for his 2001 publication, which provides a mathematical foundation to cognitive radio based on game theory. In 2005, Jeffrey became Fellow to the IEEE for contributions to software radio and communications signal processing and for leadership in engineering education.*

Virginia Tech
Wireless@Virginia Tech
439 Durham Hall
Blacksburg, VA
540-231-2972
reedjh@vt.edu
www.wireless.vt.edu

