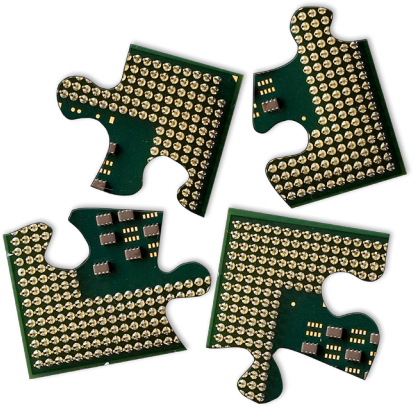
Part 2: Design and implementation of an SCA core framework for a DSP platform



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 2 of a two-part article. Part 1 ran in the March/April 2007 issue of Military Embedded Systems. *You can read Part 1 online at: www.mil-embedded.com/articles/id/?2065.*

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.

Results

Profiling was performed on the framework and applications using two different metrics: memory footprint and cycle count. All results were obtained from a single-chip configuration. That is, all framework and waveform components were collocated within the same processor. Hence, these results do not include the effects of a transport layer. *No optimization was performed* in either the framework or the waveform components and they include debug information. It is very important to emphasize that these results represent initial measurements and are subject to further investigation, validation, and optimization.

Memory footprint

The total memory used by the system is little more than 1.46 MB, which represents less than 2 percent of the available memory per DSP (128 MB) in the platform. Table 1 shows the memory breakdown by major software components. The .ERAM$heap field represents the total memory available to serve dynamic memory allocation requests. The footprint contribution from support libraries (Generic Runtime Library, Math Library, and so on) is considered under the "Other" category. Figure 4 shows a graphical representation of the main components' contribution to total memory allocation.

SW Component	Footprint (Bytes)
CF	556555
Parsers	31511
ORB	212412
Application	385624
Subtotal	**1186102**
.ERAM$heap	131072
Other	144067
Total memory	**1461241**

Table 1

In analyzing the Core Framework's (CF's) memory requirements, we found that almost 70 percent of total memory allocated for the CF came from the C++ mapping of the SCA CF IDL interfaces. It is important to note that the CF IDL descriptions contained all the interfaces defined in the SCA CF, including some that were not used in single-processor operation (for example, DeviceManager, Device). It is possible to optimize the C++ bindings of IDL interfaces by adding more control to the IDL compiler, enabling more selective code generation (such as for specific interfaces generate stub only, or skeletons only, or nothing). This approach opens the door for potentially large improvements, depending on how much of the IDL interfaces are used. This is a well-understood approach, although it was not implemented in this project. Another important qualifier for these results is the absence of Device-related interfaces. No DeviceManager or Device interfaces were implemented. The methods in DomainManager relative to Device and service registration were not implemented either.

The memory requirement results for the application include BPSK and QPSK components, along with Assembly Controller, Channel, Demodulator, Resource-

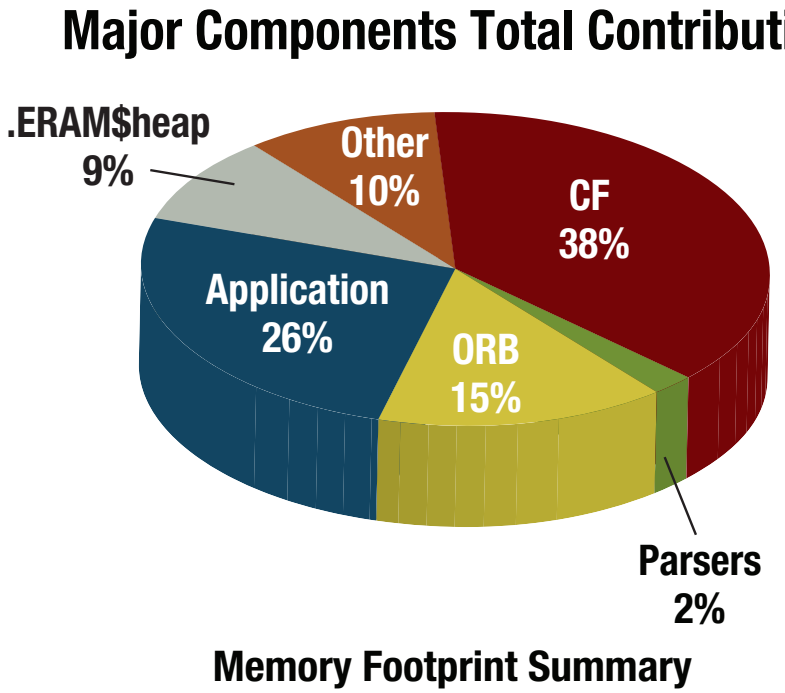


Figure 4

Factory, and the user interface. The main waveform components have very similar footprints as expected. However, the functionality of these components is extremely simple. More complex waveforms will require more memory.

Performance profile

CPU cycle requirements were collected from the CF's startup tasks: domain initialization and waveform creation. The results are shown in Table 2. Domain initialization includes the instantiation of Domain Manager, ApplicationFactory, and ResourceFactory, which durations are independent of the waveform deployed. Waveform creation represents the execution of ApplicationFactory's create(). It includes descriptor parsing, task scheduling and initialization, and component connection. Keep in mind that waveform creation is waveform-specific, and these results only apply to our test waveforms.

Task	Cycles	Time (sec) @ 720 MHz
Domain Initialization	2,365,664	3.286E-03
Create Application	10,997,946	1.527E-02

Table 2

Opposite to initialization tasks, ORB performance had a great impact on the system throughput because all intercomponent communications were established using CORBA messages. Two specific scenarios were profiled:

- **Invocation**: Round-trip cycle count for a simple method invocation with no arguments
- **Marshaling**: Round-trip cycle count for a simple method invocation with basic arguments

Two different argument types were evaluated:

- Single data type
- Sequence (1,024 elements)

In our version of e*ORB, even for interfaces with no arguments in their IDL definitions, a CORBA::Environment variable must be sent as an argument because of the lack of exception support. In both scenarios, client and server were launched as separate DSP/BIOS tasks with priorities 2 and 1, respectively. The e*ORB profiling results for different primitive data types are summarized in Tables 3 and 4.

Task	Clock Cycles
Initial Invocation	4,184
Subsequent Invs.	4,081

Table 3

	Float	Char	Short	Double
Basic Marsh.	4,208	4,127	4,142	4,124
Sequence Marsh.	6,908	5,732	6,072	8,342

Table 4

An interesting point is that the very first time a client makes a request to a server, the execution takes longer than subsequent requests, as shown in Table 3. This extra delay is because of the new connections' binding, which happens only once per connection. A graphical representation of the marshalling profile results is shown in Figure 5. After observing these results, the need for block processing in an SCA system is evident.

It takes 4,208 clock cycles to make a round-trip marshalling call with a single float, while it takes 6,908 clock cycles to send a sequence with 1,024 floats. Averaging, it only takes 6.74 clock cycles to transfer each element in the sequence.

Impact on data rate performance

The framework overhead incurred during instantiation and waveform deployment can be arranged to happen off-line. The only aspect of the SCA that impacts system throughput is the dependency on CORBA for intercomponent communications. The maximum system data rate depends on many factors: algorithm processing delays, framework delays, analog-to-digital conversion rate, and so on. To isolate the impact of the framework, we used the results shown in Table 4 to estimate an upper bound for the system data rate.

Ignoring processing delays, the maximum achievable data rate is given by:

$$R_{max} = \frac{1}{T_m + T_{tr}}$$

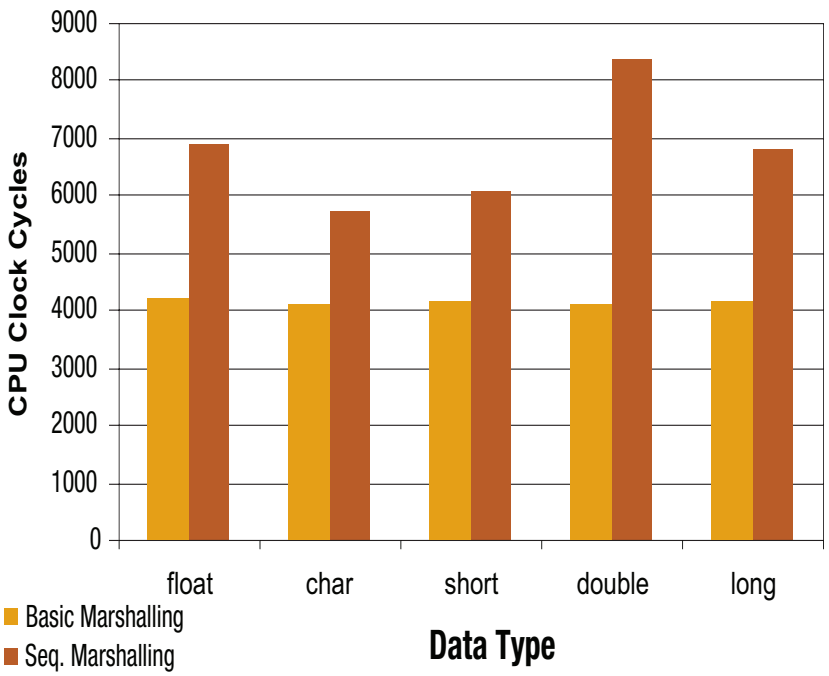


Figure 5

where T_m is the delay due to middleware processing and T_{tr} is the delay due to transport mechanisms. In our system, we only considered T_m because no transport mechanisms had been developed at the time.

The average delay per bit due to middleware message passing T_m is given by:

$$T_m = \frac{D_t \cdot S_S}{N_p \cdot n}$$

where D_t is the measured transfer delay as shown in Table 4. S_s is the number of samples per symbol. N_p is the packet size, and n is the number of bits per symbol. To estimate the maximum data rate allowed by the framework, we assumed $S_s = 8$ and $n = 1$. The clock speed in our system is 720 MHz. Substituting these values into the expression for T_m for a single float type transfer that according to Table 4 takes 4,142 cycles, the maximum data rate achievable is $R_{max} = 21{,}728$ bits per second. However, if we consider sending a sequence of 1,024 floats, the transfer takes 6,072 clock cycles allowing a maximum data rate $R_{max} = 15{,}177{,}865$ bits per second. These results highlight the need for block processing within the SCA, trading off latency and performance.

The future of SCA on DSP

One of the main concerns of following the SCA is the heavy infrastructure required to support it. To ease requirements in terms of performance, cost, and power consumption, we propose an implementation of the SCA Core Framework for a TI C64 DSP platform. This approach is feasible thanks to the latest developments in software tools and ORB technology. By having an SCA core framework in a DSP, all the benefits in software reuse and deployment flexibility brought by the SCA can be achieved in a more efficient platform. In terms of performance, our complete implementation requires about 1.5 MB, which represents about 1 percent of the total memory available per DSP in our platform. Even though CORBA introduces some delays and overhead, the overall effect can be reduced by sending data packets instead of single elements. The source code for the framework and sample waveforms is available at http://ossie.mprg.org.

Finally, it also needs to be noted that SDR technology continues to rapidly evolve and improve. The work referenced herein was conducted and profiled about a year ago. If the design and implementation were repeated today, the results (such as performance, memory footprint, and so on) would be that much more impressive in validating the SCA on DSP.

Acknowledgments

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

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