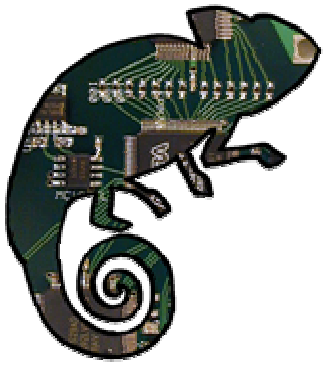

Chameleonic Radio

Technical Memo No. 3

Implementation Status of the SCA Based FM Radio

P. Balister

J.H. Reed

January 31, 2006

Bradley Dept. of Electrical & Computer Engineering
Virginia Polytechnic Institute & State University
Blacksburg, VA 24061

Implementation Status of the SCA Based FM Radio

Philip Balister and Jeffrey H. Reed

balister@vt.edu reedjh@vt.edu

January 31, 2006

This memo provides a basic description of the FM Radio architecture planned for the SCA radio and provides information about the current state of the work.

The SCA baseband processor will eventually support multichannel demodulation of narrowband FM voice and P.25 digital voice operating on an OMAP platform. The OMAP is a dual core processor developed by Texas Instruments that contains two processors, and ARM based general purpose processor (GPP) and a C55 digital signal processor (DSP) and is commonly found in cell phones and and PDA's [1].

Demonstrating two channel reception of narrowband FM voice is the first stage of several in the project. The first evaluation system scheduled for April 1 operates on PC platform using the USRP to digitize the RF from the multi-channel RF front-end. This version of the radio supports development and debugging on the radio components for the OMAP based radio.

1 Architecture for the FM Implementation

The radio software is divided into three separate areas; the SCA framework support programs, the components that provide interfaces and control for the hardware present on the radio platform, and the actual software components that perform the radio functions.

The SCA support programs are the nodebooter and the waveform loader. The nodebooter starts instances of the domain manager and device manager for the radio environment. The waveform loader provides the user interface to load a specific waveform into the SCA environment and control the waveforms execution.

Figure 1 shows the signal processing components for the radio and how they are connected to the USRP interface component and the sound card interface component. The processor proxy

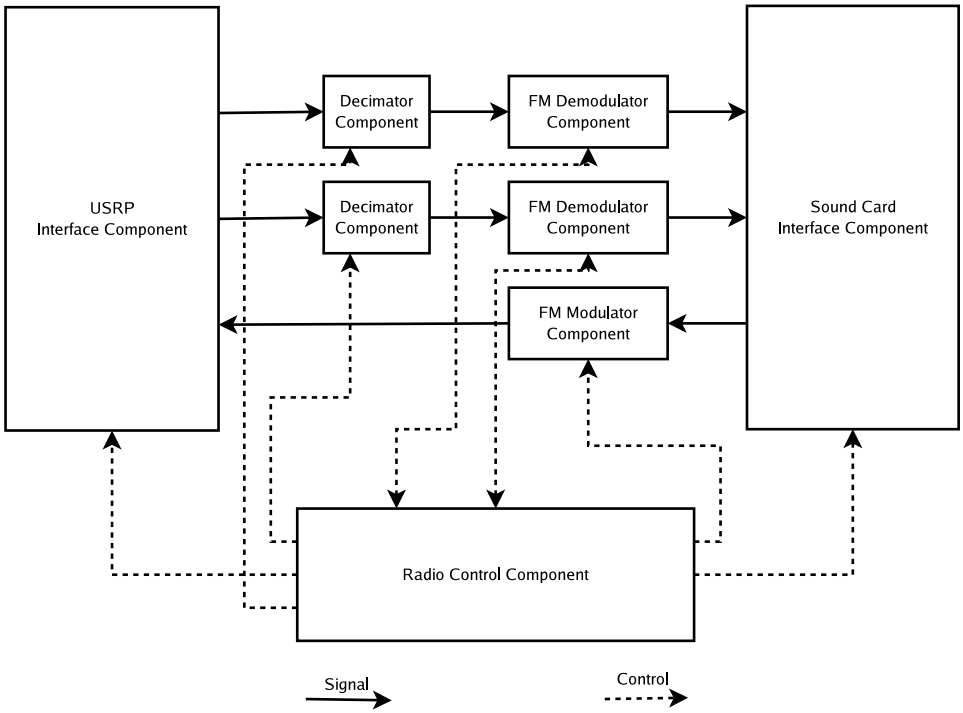

Figure 1: NIJ Radio block diagram showing CORBA connections

components are not shown in this figure since the individual components may be implemented on the GPP, the DSP, or there may be implementations available for both the GPP and DSP. The component deployment must be specified when the software is loaded into the specific radio hardware.

The hardware interface components are:

- USRP interface component
- Sound card interface component
- GPP proxy component
- DSP proxy component.

The USRP interface component provides control and data interfaces to the USRP hardware. Control information includes information such as receiver NCO tuning, receiver decimation rate, transmitter NCO tuning and transmitter interpolation rate. This component also transports the data between the USB bus interface and the CORBA interfaces supporting the actual transmitted and received data.

The sound card interfaces provide control interfaces to configure the sound hardware and provide data interfaces for the digitized sound information. The sound card device may also provide some

resampling capability to match the data rates supported by the hardware with the data rates required by the radio software components.

The GPP and DSP interface components provide proxies for deployment of the radio components. The proxies provide interfaces to execution of the radio components on the desired processing device and may provide data and control ports for components deployed on the DSP.

The radio implementation components are:

- Decimator component
- FM Demodulator component
- FM Modulator component
- P.25 Modulator component
- P.25 Demodulator component
- Radio control component.

The decimator component performs sample rate reduction and filtering. In the context of this radio, the final channel selection occurs here. The USRP stock firmware can only decimate to a final sample rate of 250 Ksps. For narrowband FM voice the channel required channel widths are 12.5 kHz or 25 kHz. The software decimators will reduce the sample rate to a rate that supports demodulation of the FM signal and the filter will provide the required channel width. Since the USRP decimation is applied to both received channels, selection of 15 kHz versus 25 kHz is made by this component.

The FM demodulator receives the channelized signal from the decimator component. Using a phase locked loop detector (PLL) the baseband voice data is recovered from the FM signal. Before sending the sound data to the sound card, filtering will remove the pre-emphasis applied to the signal. This component also provides detection of any required squelch system based on the sub-carriers embedded in the transmitted signal.

The FM modulator receives the digitized audio from the sound card. When the modulator receives the push to talk signal from the radio control component, the audio is used to create a narrowband FM signal based on data from the radio control. Both 12.5 kHz and 25 kHz modes are supported. The audio signal must be bandlimited to 3kHz and have pre-emphasis applied to the signal to improve the system signal to noise ratio.

The radio control component serves as the SCA assembly controller component. The assembly controller designation means the framework sends waveform control messages to this component. The control component also provides the required user interface, such as frequency selection, frequency display, and a push to talk button. Finally, this component sends setup information to the other radio components based on the desired operating channels and monitors the incoming channels to provide the user with active channel feedback.

Work on the P.25 digital voice format starts in June.

2 Status

Before starting the FM Radio design, the OSSIE framework was upgraded to support an improved component model for waveform implementation. These upgrades enhanced the component based nature of the framework and allow easier re-use of components. These improvements included developing a waveform independent nodebooter and waveform loader user interface program.

The GPP proxy component is complete. DSP proxy component development start on April 2. Some redesign of the GPP proxy component may occur based on the evolution of the DSP component design.

The USRP interface device has been tested in receive mode. No obstacles are anticipated implementing the corresponding transmit software.

The skeleton software for the FM portion of the radio has been developed and compiled. This software is currently under test.

One key design decision remains. Currently the FM demodulator creates data with a sample rate determined by the incoming channel width. Since the modulator input sample rate is larger than the rate required for the output audio signal bandwidth of 3kHz [2], sample rate matching is required before the data is sent to the actual sound hardware. The location of the final sample rate conversion is not clear at this time. The final determination will be made after the sound interface device is complete and we understand the available sound playback rates available on the hardware available for the project.

Although the immediate goal, the PC based radio system, does not require use of the OMAP system, some of the PC based work has been compiled and run on the embedded system. One of the benefits of the Open Embedded system described in Appendix A [3] is that it is simple to verify compilation of the PC platform software for the OMAP processor environment. Appendix A also

describes OSSIE, which is the SCA framework our FM radio is based on. Currently, the OMAP platforms runs the nodebooter process, the GPP Proxy device, and the USRP interface device. As new components are completed on the PC platform, limited testing on the OMAP platform will performed to quickly uncover any problems that may arise.

References

[1] http://www.ti.com/omap.

[2] TIA, *Land Mobile FM or PM - Communications Equipment - Measurement and Performance Standards, TIA-603-C*, December 2004.

[3] J. H. Reed, M. Robert, and P. Balister, "Embedded Software Defined Radio (SDR) with the OSSIE Core Framework," tech. rep., MPRG, 2005.

Appendix A
Embedded Software Defined Radio (SDR) with the OSSIE Core Framework

Dr. Max Robert Philip Balister Dr. Jeffrey H. Reed

July 27, 2005

Abstract

Software-Defined Radio (SDR) systems require a software architecture to support realtime reconfigurability and radio flexibility. One such architecture is the SCA (Software Communications Architecture), the architecture used by the Joint Tactical Radio System (JTRS) program. This paper describes the porting of an implementation of the SCA, OSSIE (Open Source SCA Implementation Embedded), to an embedded environment. OSSIE is an open source C++ implementation of the SCA specifications that was implemented under the DCI Postdoctoral Research Fellowship. The target embedded platform for this study is the OMAP processor, a dual-core Texas Instruments processor. After the successful port of the framework and supporting software to this platform, a test waveform was implemented and evaluated. The waveform implementation required the addition of swap space in the form of a micro-drive to function properly. Information collected during the evaluation process provides a path that can be followed to minimize the overall memory footprint of SCA-based system on embedded platforms.

1 Introduction

As part of the DCI Postdoctoral Research Fellowship program, developers at Virginia Tech created an open-source C++ version of the Joint Tactical Radio System (JTRS) [1] Software Communications Architecture (SCA). This open-source software is called OSSIE (Open Source SCA Implementation Embedded) [2]. The SCA is configuration management software that is intended to manage multi-band multi-mode software-defined radio systems. In this context, a software-defined radio (SDR) system can be defined as one that can significantly change its functionality through only software changes. For example, a typical cellular phone is designed to operate in the 800 MHz and 1900 Mhz bands at the RF level and to support a couple of cellular telephony standards such as GSM and WCDMA. The cellular telephony standard support is generally done through a combination of programmable devices and application-specific integrated circuits (ASICs). If one were to change this cellular phone to a software-defined radio system (and it included hardware with sufficient computational power), it would be able to support not only GSM and WCDMA, but also other standards such as IS-95, CDMA-2000, and iDEN, since the standard it supports would be defined through software. This example can be extended with a flexible RF front end that can support signals extending to other bands, such as the 2.4 GHz ISM (Industrial Scientific Medical) band, and there it may support standards such as Bluetooth, 802.11b (Wi-Fi), and 802.11g. The SCA is designed to support this type of functionality as well as supporting code re-use, thus reducing the development cycle of a system.

2 The SCA

A thorough description of the SCA is beyond the scope of this paper. However, a brief overview of the specification is included to provide a point of reference to the reader. The SCA is system management software. Its goal is to control processing and RF hardware, manage the deployment of software on the available hardware, maintain a unified file structure, and maintain a consistent component control structure. In short, the SCA does all that an OS does except process and thread management. Beyond this functional difference, the key difference between the SCA and an OS is that the SCA is designed to support distributed hybrid systems.

The SCA can be described in four basic pieces: Framework services, component services, profiles, and interfaces. Framework services are the general services that are expected from any management software, namely the ability to launch and terminate waveforms, such as GSM or IS-95, install and uninstall hardware devices, and maintain a distributed file system. Component services are the basic services necessary for a component to operate. These services include providing interfaces for the waveform creator to connect components, constructors and destructors, and component properties. Profiles are probably some of the most novel aspects of the SCA. Profiles are a collection of XML (eXtensible Markup Language) files that are used to describe hardware configurations, system configurations, component structures, and waveform content. Finally the interfaces are the collection of APIs (Application Programming Interfaces) that standardize the interface between different components. The use of APIs is critical to maintain the ability to reuse code.

At runtime, an SCA radio would have some bootup software which installs the available hardware and storage devices onto the framework. Once the bootup sequence is complete, all the available hardware is installed on the system, the middleware (more on this later) is initialized, and a list of the available waveforms is created. At that point, the framework is ready to respond either to a user request to install a particular waveform or reacts to some other software instructing which waveform should be installed. When a waveform is installed, the different functional components making up the waveform are instantiated by the framework and connected. At that point, the waveform is operational and providing services to the user.

3 OSSIE

One of the biggest problems with the SCA is availability of code. Available frameworks are either too expensive or not sufficiently simple to support basic SDR research. OSSIE, and implementation of the SCA implemented under the DCI Postdoctoral Research Fellowship Program, was initially designed to run on desktop computers with the Windows 2000/XP operating system. These machines generally have hundreds of MB or more of memory capacity as well as tens of GB of fixed storage (hard drives). Furthermore, new versions of these machines have processing cores that operate at speeds in excess of 3 GHz and are plugged directly into the power outlet, making power consumption irrelevant. As part of the DCI Postdoctoral Research Fellowship work, it was deemed important to evaluate the suitability of the SCA to embedded environments. Such environments are usually heavily constrained in memory availability, long-term storage availability, and power supply.

Wireless communications places constraints on processors that go beyond traditional computer applications. A wireless system's baseband processing is composed of well-known signal processing blocks, thus allowing the use of processors with specialized instruction sets. Given this constraint, it was deemed important to use a processor that had the infrastructure necessary to support the SCA, namely an operating system and an implementation of CORBA (more on this in the next section), as well as the appropriate instruction set for wireless communications systems. Given these constraints, The Texas Instruments OMAP 5912 processor provides an interesting platform for implementing a SCA compliant radio [3]. This platform provides an ARM 926EJ-S processing

Figure 1: OMAP 5912 OSK

core running at 192 MHz with a C55 DSP core also running at 192 MHz [4]. The ARM core is a low-power RISC processor that can support the needed software infrastructure to support OSSIE as well as a signal processing core designed specifically for wireless communications. The OMAP 5912 that was used in this implementation is the one available in the OMAP 5912 OSK (OMAP Starter Kit), a board that contains an OMAP processor as well external interfaces, such as a USB port and an interface for industry standard Compact Flash memory. Figure 1 shows a picture of the OSK. The configuration of the board allows for 32 MB of on-chip memory. Additional memory is available through the Compact Flash interface, although significant performance penalties are incurred when accessing storage through this external interface.

This paper discusses the challenges that arose from the port of OSSIE from a high-power general-purpose processing platform to the resource-limited OMAP 5912. Furthermore, a simple waveform was created to test the capabilities of this port. The waveform collects data from an external device, the USRP (Universal Software Radio Peripheral) [5], a four-channel IF interface with some digital up- and down-converters and a USB interface for data transport, and passes it to the ARM core for display. The use of the C55 core is not discussed on this paper, since the goal of this paper is to look at the overhead incurred by the SCA on a system, and given the nature of the SCA, that overhead is limited to the ARM core.

4 Underlying Software

The SCA is not designed to stand alone. Instead, it leverages a wide variety of software packages. As seen in Figure 2, the management software relies on a central general-purpose processor, which runs an operating system (OS). The role of the operating system is to support functionality such as process and thread management, file management, and device drivers. The operating system provides a common interface for that specific programmable core, thus simplifying the task of the developer. The SCA calls for the use of a layer of middleware above the OS. This layer provides the environment with the ability to access other functional blocks in a distributed environment. In a single-core environment, such a layer is not generally needed, since the operating system performs the needed functionality. However, in more complex systems requiring more than one core, such a layer greatly simplifies development.

In the case of the SCA, this additional middleware is CORBA (Common Object Request Broker Architecture) [6]. CORBA is relatively sophisticated software designed for large-scale distributed environments. However, in the case of the SCA, a sizable slice of CORBA functionality is not necessary. Instead, CORBA is used to perform a small set of operations, namely manage remote pointers for objects, marshal information, and transport information. The CORBA functionality

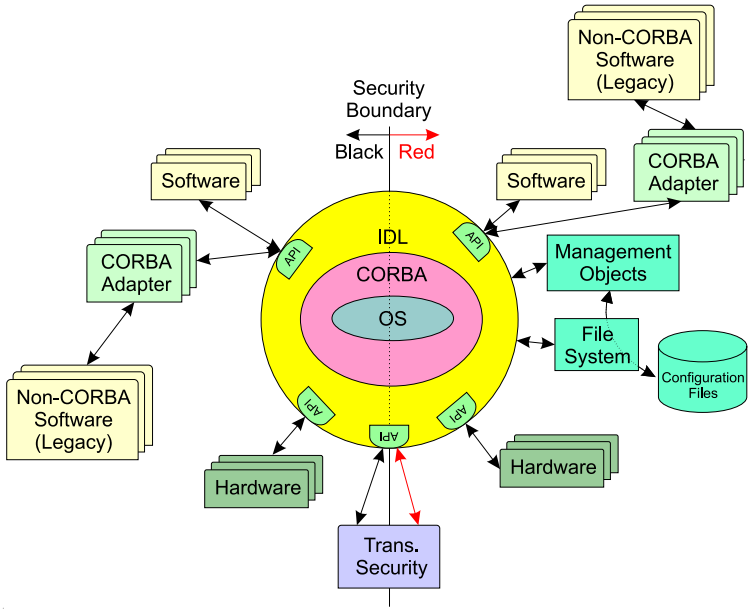

Figure 2: Overall Structure of SCA Software

needed for the SCA is a slightly more than that available from basic RPC (Remote Procedure Calls) functionality. One of the appealing properties of CORBA is that it allows the developer significant latitude in his choice of development language. To provide language independence, CORBA includes a language called IDL (Interface Description Language). IDL is used to describe the relevant interfaces of an object. Once these interfaces have been described in IDL, a code generator, which is provided by the CORBA vendor, is used to generate interface code in the developer's language of choice. The developer's only limitation in language choice is the set of languages that the CORBA vendor chose to select in their IDL code generator; there is no intrinsic limit within IDL. Once this interface code has been created, the developer can then add the generated files to the build. These generated files provide all the interface necessary to use CORBA. The interface capabilities that IDL provides are represented as an abstraction layer in Figure 2.

Beyond these underlying layers of software, additional software such as a development tool chain is required. While the development tool chain is not a run-time issue like the choice of OS or CORBA implementation, it has a critical determinant for the selection of the underlying software.

5 The Port

This section discusses the issues and choices at the different stages of the development process. First, the development environment is discussed. In the second section, the CORBA middleware selection is discussed. In the third section, there is a brief discussion of the data acquisition package. The fourth section discusses waveform development. Finally, benchmarks on system performance are provided.

5.1 Software Development Environment

In this context, the software development environment (SDE) comprises the choice of operating system and associated tool chain. The software development process for an embedded system fundamentally differs from the software development process for a desktop system; due to expected deployment constraints, an embedded system does not typically have the resources to support a software development environment including editors, compilers and libraries. Therefore, when

searching for an embedded operating system, the available tool chain, which is used for cross-compilation onto the target platform, is critical.

A preview of the Monta Vista Linux development environment is supplied with the OSK [7]. This package includes the tools required to develop software for the OSK in a convenient environment with technical support available from Monta Vista. Several issues arose with Monta Vista, namely compatibility with development platforms and kernel compatibility for long-term peripheral support. A review of alternative development systems led to the Openembedded (OE) project. OE is a development environment designed to produce file system images for embedded Linux hardware. While OE has a significant learning curve, it is fairly easy to use once setup. The output of OE is an image of the filesystem that is installed on the embedded hardware. The developer selects a package set that is to be deployed on the embedded device. With this selection, OE then automatically finds and builds all dependencies of the selected package, yielding the desired build. OE is designed for such embedded projects and supports several different hardware platforms capable of running Linux. A typical OE target is a handheld device such as a Sharp Zaurus [8] or a HP iPAQ [9]. The relative complexity of OE means that it has a significant learning curve when developing large packages. However, users that do not do system-level development (OS, middleware) can quickly learn to use the environment.

The initial download of OE already supports the basic system libraries and utilities required to operate the OSK, so effort was focused on adding support to OE for the OSK machine definition, the XML parser (for the profiles), a CORBA implementation, and the support software for the data acquisition board.

5.2 OSSIE Framework porting

OE provides the basic compiler tools and some of the libraries necessary for OSSIE. The missing libraries are the XML parser and CORBA. The initial version of OSSIE for the desktop computer used Xerces as the XML parser and TAO as the CORBA implementation. Xerces is an open-source XML parser that is available from the Apache project [10]. Xerces is available for both Windows 2000/XP and Linux, making it a desirable choice for OSSIE. TAO (The ACE ORB) is an open-source implementation of CORBA that is available for a large variety of platforms [11]. Unfortunately OE does not include build files for the Xerces XML parser nor the TAO CORBA implementation.

The build files necessary to add Xerces support within the OE were simple and quickly added. TAO, on the other hand, proved to be not only memory-heavy but also difficult to use with OE. Because of the nature of the OSSIE project, it is important for different versions to diverge as little as possible. Therefore, an ORB was needed that supports the minimum CORBA requirements and runs on Linux and Windows 2000/XP and is available as open source. OmniORB was identified as a suitable replacement for TAO; omniORB supports the minimum CORBA requirements, is available for both Linux and Windows, is simple to install, and is optimized to generate software with a smaller footprint than TAO [12].

The OSSIE framework required some additional modifications to compile successfully with omniORB. Beyond the changes in library includes, there were some changes required when the framework used TAO-specific constructions to interact with CORBA. These instances were changed to match the CORBA specifications where possible. This should improve OSSIE's ability to work with other CORBA packages if it were ever required.

5.3 External interface

The external interface used is the Universal Software Radio Peripheral (USRP) is an interface board containing four analog to digital converters (ADC) and four digital to analog converters (DAC), thus the USRP can theoretically support a four-channel transceiver [5]. The ADC's have a sampling

rate of 64MSps with a 12-bit resolution, allowing the acquisition of up to 32 MHz of bandwidth. The DAC's have a sampling rate of 128MSps with a 14-bit resolution, making it a reasonable match to the ADC side of the board. Up- and down-conversion is performed through CIC filters implemented on an on-board Altera EP1C12 Q240C8 "Cyclone" FPGA. The interface between the USRP and the processing hardware is through a USB 2.0 line, which can support a maximum of 32MBps. Given USB's maximum bandwidth and assuming 16-bit IQ pairs for each sample, then the maximum aggregate bandwidth that the USRP can support is 6MHz. Unfortunately, the OSK only supports USB 1.0, with a maximum transfer rate of 1.5MBps, further reducing the maximum supported bandwidth to approximately 280kHz. Even though the maximum supported bandwidth is relatively narrow, this configuration does provide a relatively simple platform on which concepts can be evaluated. It should be noted that the original USRP does not contain USB 1.0 support, and the board's firmware had to be modified to support this older version of the standard.

5.4 Test waveform

The framework does not stand by itself - it is used within the context of a waveform (application). In this case, the waveform is a simple three-component one. The first component is one that is called modulator, and receives data from the USRP, which is configured to receive data centered around 1.5 MHz and decimates the sample rate by a factor of 256. The second component is called channel, and acts as an all-pass filter. Its only role in this waveform is to create an additional (artificial) burden on the infrastructure. The final component is called demodulator, and for every packet of samples received, it calculates the packet energy by squaring each sample and summing the resulting squares. This waveform demonstrates some basic functionality, receiving data from an external source and processing the data with an algorithm suitable for execution on a CPU that only supports integer math, like the ARM. It should be noted that at this stage, the OMAP's C55 core is not being used.

6 System memory performance

Given the multiple layers of software that are used by an SCA radio, one key limiting factor for such systems is the amount of physical memory available to the processor. The OSK board used for this system has 32 MB of RAM. As mentioned previously, the SCA requires support from a CORBA package and an XML package. Both of these packages place significant demands on available system memory beyond those traditionally placed on an embedded system. The OMAP processor contains a memory management unit (MMU) that is used by the Linux operating system's virtual memory subsystem to effectively manage the process address space use of physical memory [13].

In general, embedded systems do not use shared libraries. There are several reasons for this choice. In general, dynamic libraries require symbols to be included in the executable code for runtime linking of the library. This additional code makes the executable footprint larger. Furthermore, because a runtime load is required, not only is this process slower, but it increases the level of uncertainty concerning the runtime performance of the code. Finally, some embedded systems do not use an MMU, and thus code that uses dynamic libraries can be more complicated. However, in this case, the SCA uses the same linked code in a variety of concurrent executables, and a shared library approach should be more efficient. All major libraries, i.e.: CORBA, XML, OSSIE framework, are compiled as shared libraries, reducing the overall memory footprint to attempt to fit within the tight bounds of the OSK board.

The test waveform developed for this project would not run suceesfully in 32MB of RAM without swap space. By adding a swapfile on a micro drive mounted in the compact flash slot, the system could free physical memory by writing data to the swap file as required, freeing sufficient memory to process all aspects of the waveform start up. Once the waveform is loaded and starts running,

much of the code in the support libraries is no longer required, this allows the VM system to use physical memory only for code that executed during waveform processing. At this stage, access to the swap file is minized and the system runs more efficiently. Benchmarking suggests further optimization of the OSSIE framework and supporting libraries may be required to further address the needs of the embedded environment.

7 Conclusion

In this paper, a port of an implementation of the SCA (Software Communications Architecture) to an embedded environment was described. The SCA implementation selected is OSSIE (Open Source SCA Implementation Embedded), an open source C++ implementation of the SCA specifications that was implemented under the DCI Postdoctoral Research Fellowship Program. The target embedded platform for this study is the OMAP processor, a Texas Instruments processor with two cores, an ARM 926EJ-S and a C55 DSP. Porting steps included the selection of a more efficient CORBA implementation, omniORB, and the selection of an embedded Linux variant available through Open Embedded. The OSSIE framework had to undergo some modifications in the porting process. After the successful port of the framework, a simple test waveform was implemented and evaluated. To function properly, a simple waveform implementation required the addition of swap space, in the form of a micro drive. As part of the evaluation process, it became apparent which libraries occupied the most memory on the system. This additional information provides a path that can be followed to minimize the overall memory footprint of SCA-based system on embedded platforms.

8 Acknowledgements

This work was sponsored by the DCI Postdoctoral Research Fellowship and the MPRG Affiliates Program

References

[1] http://jtrs.army.mil/.

[2] http://ossie.mprg.org/.

[3] http://www.ti.com/.

[4] http://focus.ti.com/docs/prod/folders/print/omap5912.html.

[5] http://home.ettus.com/usrp/usrp_guide.html.

[6] http://www.corba.org/.

[7] http://www.mvista.com/.

[8] http://www.dynamism.com/zaurus/.

[9] http://welcome.hp.com/country/us/en/prodserv/handheld.html.

[10] http://xerces.apache.org/.

[11] http://www.cs.wustl.edu/ schmidt/TAO.html.

[12] http://omniorb.sourceforge.net/.

[13] R. Love, *Linux Kernel Development*. Novell Press, 2 ed., 2005.