TECHNIQUES AND RECOMMENDATIONS TO IMPROVE WAVEFORM PORTABILITY

Scott Macejak, David Maldonado, Jim Agniel
(L-3 Communications Nova Engineering, Cincinnati, OH, USA;
Scott.Macejak@…, David.Maldonado@…, Jim.Agniel@L-3Com.com)

ABSTRACT

With the creation of the Software Communications Architecture (SCA) as part of the Joint Tactical Radio Systems (JTRS) program, the government has invested substantially to address the need for waveform portability. However, to date, the SCA has emphasized the software residing on a general purpose processor (GPP). The SCA specification does not adequately address the physical layer (PHY) waveform processing as the PHY implementation is typically accomplished in digital signal processors (DSPs) and field programmable gate array devices (FPGAs). A 2006 GAO assessment of the JTRS program cited unacceptably high porting costs and long porting schedules, indicating that the JTRS waveforms have yet to live up to the objective of highly portable, hardware-agnostic software applications. In this paper, however, we show through two case studies that the portability goals of the JTRS program are achievable. We address the importance of the waveform software architecture and the role of a Portability Toolkit in reducing waveform porting costs. We also identify the non-portable techniques that currently prevail in the industry.

The recommendations presented here derive from our experiences with the Single Channel Ground-Air Radio System (SINCGARS) waveform, Wideband Network Waveform (WNW) OFDM PHY, and Soldier Radio Waveform (SRW).

1. INTRODUCTION

Past experience in both waveform development and waveform porting has taught us an important lesson: Porting non-portable code costs more than re-writing the code from scratch. With this premise in mind, the continued development of non-portable code is undermining key goals of the JTRS programs. This conclusion is consistent with the performance described by the GAO in its 2006 review of the JTRS program [1]. To address these concerns, the JTRS Network Enterprise Domain Test and Evaluation (NED T&E) created a set of Waveform Porting Guidelines [2]. We endorse these guidelines and encourage their proliferation. This paper provides further recommendations to improve waveform portability, with the intention of reducing the overall cost of delivering SDR waveforms into JTRS programs.

For the purposes of this paper, we define *portability* as the degree to which the cost of porting a waveform is minimized. Ideally, waveform code from the JTRS repository should remain unchanged when utilized in different hardware platforms, and the porting effort should be relegated to configuring hardware devices, interfacing to a hardware abstraction layer (HAL) and performing system integration, test and validation. However, this is often not the case—waveform specific code modifications that are required because of limited portability can cause significant unforeseen costs when porting to new platforms. To address these cases, we present a methodology and tools that enable a straightforward and successful porting effort.

2. SDR SOFTWARE ARCHITECTURES

SDR software architectures are typically established by partitioning waveform control and signal processing functions among a platform's available hardware resources. The software architecture takes into account the capabilities and resources of the devices available on a given target platform.

We characterize SDR software components as either *real time* or *non-real time* elements. Real time functions such as high rate and/or computationally intensive signal processing are more appropriately partitioned to a DSP or FPGA, while non-real time functions, such as control signaling and low rate signal processing, are optimally implemented in a microprocessor or DSP. FPGAs, DSPs and GPPs are found in most SDRs, and software architects often have a degree of flexibility when adapting waveform software to a particular platform, depending on the platform's capabilities and the processing demands of the waveform components.

As waveform software is ported to various platforms, developers may need to modify the software architecture to accommodate different hardware architectures. However, herein lies a central problem: Should a given SDR waveform implementation be expected to run on any SDR

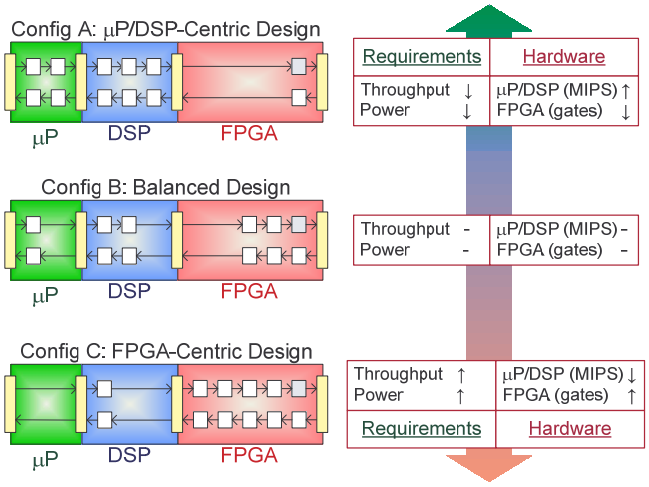

Figure 1: The radio platform's processing resources may require that the waveform software be repartitioned to meet performance specifications.

hardware platform in order to be considered "portable"? While the ideal portable waveform would run on any radio platform, we allow that the practical answer is *no*. However, the standard of waveform portability we promote in this paper requires that waveform software be supported with several, specific portability artifacts, which we define in the next section. Section 5 describes two porting efforts that were supported with Toolkit artifacts, including successful waveform software repartitioning, at minimal cost.

3. THE PORTABILITY TOOLKIT

The artifacts typically available to developers when launching a porting effort are generally inadequate to enable cost effective waveform porting. Often the primary (sometimes the only) artifact delivered to a waveform porting team is source code. Source code, no matter how thoughtfully designed, is not sufficient to realize a truly portable waveform or to port waveform software efficiently. Source code has often been optimized for a particular platform or device, can be hard to read and does not provide enough information for effective debugging. In fact, source code alone is of limited importance, and should be only one component of a waveform's *Portability Toolkit*. A Portability Toolkit should be made available for each waveform. Aside from well documented object-oriented source code, L-3 Nova regularly creates a set of portability assets to streamline the porting process. At a minimum, we recommend that an industry standard Portability Toolkit include the following four artifacts: 1. detailed system, software and design documentation, 2. a non-real time, PC-based emulator, 3. full, functional and bit-true waveform behavioral models and simulations in MATLAB, Simulink and/or OPNET, 4. testbenches and test vectors at both the component level and top level.

3.1 Waveform Documentation

Our experience has led us to the conclusion that the detail provided in waveform documentation is often insufficient to be effective in waveform porting as the documents rarely have the necessary detail for efficient debugging. In the case of C++, detailed class diagrams, intended multi-threading scheme and comprehensive unified modeling language (UML) sequence diagrams would go a long way in painting the overall picture that is generally missing when just looking at source code. In the case of the VHDL, diagrams of the clocking scheme, detailed block diagrams and RTL documentation for each of the primary components would be especially helpful.

3.2 Non-Real Time (NRT) Environment

When porting code to a new platform, developers require a simple, flexible environment for testing and debugging delivered source code independent of the hardware. For these purposes, it is generally not critical that the simulation environment be capable of operating the waveform in real time. Instead, key features of the emulator should include the ability to trace execution paths through the code, to halt code execution arbitrarily, to inspect the internal state of variables and to output useful information to a logging device or screen. These features have all become common in modern debugging applications and make a PC an excellent choice to host the NRT Environment.

In addition, the NRT Environment provides simulated interfaces into the networking layer, the physical layer and the operating environment of the platform. It also provides high visibility into the waveform operation that is often lacking in a hardware configuration; the emulator acts as a pre-integration step to discover problems that are difficult to capture in hardware. Depending upon the requirements of the waveform, OPNET models can also be made to encompass the NRT. L-3 Nova's development of an NRT Environment for the JTRS SINCGARS waveform was a key factor in the success of the porting effort for the GMR program.

3.3 Waveform Modeling and Simulation Code

Another important artifact in the Portability Toolkit is the set of MATLAB/Simulink models. These models enable the implementation to be readily and easily modified, re-architected and recreated, if need be. Additionally, the models provide a method of independent verification of the implementation. These models are also far easier to read and understand than embedded C++ or VHDL. The MATLAB/Simulink fixed point models need to be accurate

to the degree that internal test vectors can be generated for any point in the signal processing chain.

While the MATLAB model and NRT Environment validate the system in a point to point environment, an event-driven model from a tool such as OPNET enables the waveform to be validated using networks of multiple participants. This verification is especially critical for networked waveforms such as WNW and SRW. The use of OPNET models also greatly reduces the risk of field testing as it allows testing of network scenarios that cannot be tested in a lab, such as large networks or networks with high mobility. For maximum effectiveness, the OPNET model should use a large percentage of the actual waveform code, rather than code that simply models the waveform behavior. Since the embedded code base and the OPNET model are highly synchronized, problems can be identified and resolved in both a lab environment as well as in an OPNET simulation.

3.4 Software Testbenches

Component level unit testbenches should bridge the design and implementation, enabling verification that the waveform implementation behaves as designed. Creating test artifacts in MATLAB/Simulink, C++ and VHDL provides the means to debug the design at its lowest level. This makes it far easier to isolate and debug issues at the component level. Designers may or may not create these assets on a case-by-case basis, but when delivering portable waveform software, these test assets are critical to enabling a low-cost port to the next target platform.

Although at first look it may seem too costly to provide these recommended Portability Toolkit artifacts, our experience has repeatedly demonstrated the high value of the Portability Toolkit in reducing the cost of subsequent waveform ports. We routinely develop each of these artifacts during an initial waveform development, as well as at the first port of a new waveform, and we find that the generation of these assets typically leads to a lower overall development and porting cost via a shortened system integration and debug cycle. Additionally, the Portability Toolkit can enhance (if not guarantee) interoperability among different hardware platforms running the same waveform. From the government's point of view, the total lifetime cost of the waveform is further minimized, since the Portability Toolkit ensures minimal cost of porting waveforms to future platforms.

4. SOFTWARE DESIGN TECHNIQUES TO AVOID

Avoiding design techniques that reduce portability is important to minimizing waveform porting costs. Some portability-enhancing techniques are "best practices" and the industry standard, such as the use of classes in C++ and generics in VHDL. We present here some software practices that are widely used in industry because of their promise to reduce design cycle time, but that actually have the effect of reducing portability.

4.1 Platform-Specific Optimization

Whether code is targeted for a GPP, DSP or FPGA, a good indicator that a non-portable design has been developed is evident whenever a high degree of effort performing platform-specific optimization is required to implement a waveform. Platform-specific optimization does not refer to good DSP design methods such as memory management and/or object-orientated design or good FPGA design practices such as pipelining to improve timing and folding to reduce FPGA resource footprint.

For processors, if complex optimization settings must be applied to achieve the required performance and correct behavior, it will be difficult to reliably reproduce the same results in another device. The effects of optimization can vary between devices and compilers, making it sometimes impossible to replicate the results of the previous platform. This can require a code rewrite and possibly a design re-architecture, turning a porting effort into a redesign effort and greatly increasing the amount of time required.

For FPGAs, platform-specific optimization has several indicators. Lack of portability is indicated if the use of physical floorplanning or the use of non-default synthesis and place-and-route options are required to meet timing constraints. These optimizations do not translate well over to other FPGA vendors' devices and usually do not translate well even to other families within the same FPGA vendor.

4.2 Use of Intellectual Property (IP)

Use of IP is prolific in the industry and provides for improved time to market. However, certain types of IP should be avoided. To be portable, designs must not utilize IP cores specific to an FPGA vendor. This seems obvious, yet there are a plethora of cases of this design practice being utilized.

There are two sub-categories to this practice: IP functions and IP resources. An IP function is defined as a stand-alone block generally used to implement traditional signal processing in which its source code is "hidden" from the developer. The inner workings of the IP function is unknown and the code itself cannot be changed beyond parameterization. This presents a problem because the function cannot be altered or optimized to support changes to the waveform implementation. This is especially problematic if the function is specific to a certain FPGA vendor such that a new IP function must be created if moved to another FPGA vendor's device.

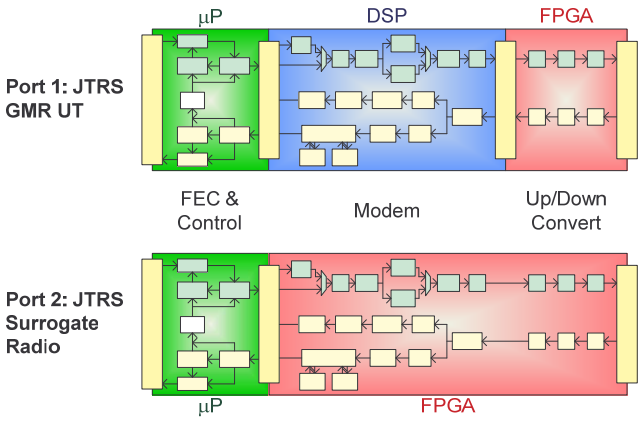

Figure 2: Two examples of L-3 Nova's JTRS SINCGARS waveform partitioning.

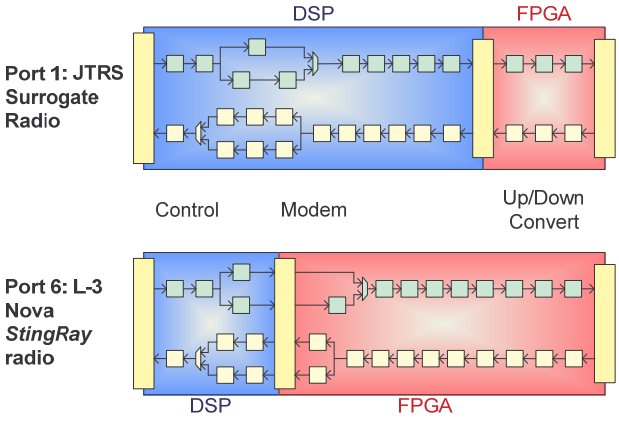

Figure 3: The WNW-OFDM waveform partitioning has evolved to accommodate different hardware architectures. Port numbers reflect the numbering scheme in Table 2.

If the complexity of the function is low, such as FIR filters, FFTs, or AES encryption, it is easily within a waveform developer's capabilities to create these functions in traditional VHDL. An exception case may be warranted for highly complex functions such as Turbo or LDPC FECs; however, third party developers will often have VHDL cores available as an alternative. SRW is an example of a waveform that uses a proprietary FPGA core to implement critical signal processing functionality. The use of a proprietary core is one of the reasons that SRW is not very portable.

The second sub-category to the IP core practice is the use of IP resources. An IP resource is defined as a FPGA-specific structure such as a multiplier or embedded RAM block. It requires the waveform developer to utilize a tool specifically developed by the FPGA vendor to create the IP resource; the resulting resource is specific to that FPGA vendor's device. Therefore, porting to other FPGAs forces the developer to re-create or replace every IP resource.

The additional danger of using IP resources is that equivalent structures must exist in the other FPGA devices. For example, the asynchronous RAM block utilized in the Ground Mobile Radios (GMR) Modem HAL (MHAL) has no analogue in Xilinx FPGAs. When we ported the MHAL to the JTRS Surrogate Radio, there was not an asynchronous RAM IP resource in the device. Since the MHAL functionality depended on the availability of this particular resource, the MHAL had to be significantly re-designed to work with synchronous RAM blocks instead.

Inferring IP resources has added advantages in being able to optimize for a platform's resources. For example, consider a platform with limited memory resources. Read-only memories (ROMs) are often generated to store trigonometric look-up tables. Using standard IEEE packages, it is simple to create a user-defined sine/cosine ROM that has generically configurable addresses and data widths in portable VHDL. Therefore, if necessary to accommodate a platform with limited memory resources, the design can be made smaller by changing one or two generics.

Today's synthesis tools have the capability to infer IP resources. Code written in generic VHDL is automatically targeted to the device's specialized IP resources. The Synplify synthesis tool is convenient in that it supports resource inference for several vendors' FPGAs, although other tools such as Altera's Quartus and Xilinx's ISE support inference as well. A simple test that L-3 Nova performs to validate portability is to use the Synplify tool to synthesize a waveform implementation to several device families of both Altera and Xilinx products.

4.3 Multiple Clock Domains

It has long been good design practice to produce FPGA designs with as few clock domains as possible. In designs targeted for low power operation clocking the logic at the lowest possible rate does reduce the power consumption. However, this approach can negatively impact portability.

SRW is an instance of a waveform developed for low power operation which utilizes multiple clock domains within the modem. Their use becomes a problem when hosted on a platform that is unable to generate the required number of clocks. While not always feasible, the practice of using more than one or two clock domains should be avoided.

5. WAVEFORM PORTABILITY CASE STUDIES

Two JTRS waveform porting case studies, shown in Figures 2 and 3 above, illustrate the importance of the Portability Toolkit and substantiate our recommendations for portable software techniques.

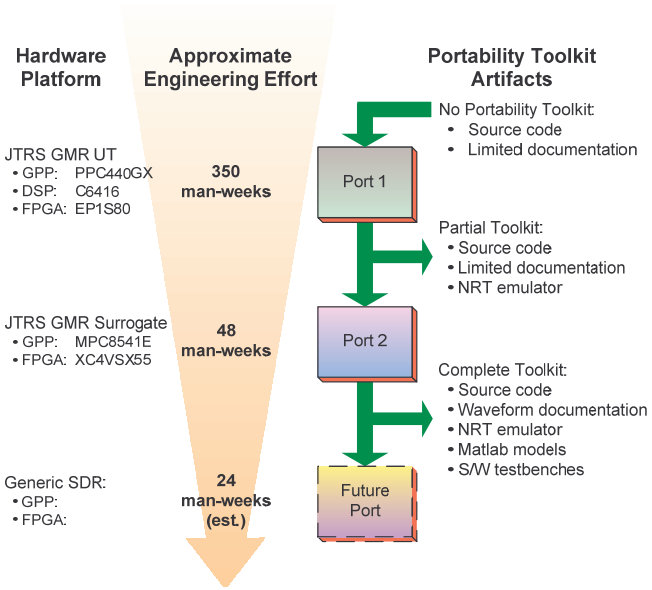

Figure 4: The JTRS SINCGARS Waveform porting process improved as Portability Toolkit artifacts were created.

5.1 Case Study 1: JTRS SINCGARS

This case study is based on two ports of the SINCGARS waveform: an initial port to the GMR platform and a subsequent port to a COTS GMR surrogate platform. Both ports included single channel and frequency hopping SDM SINCGARS modes, including both voice and 16K data. Crypto support, RF integration and EDM mode are not considered in this case study. Figure 4 outlines the porting processes and identifies maturing artifacts that resulted.

The original JTRS SINCGARS waveform development, the output of which served as the baseline for the JTRS GMR SINCGARS port, resulted in a GPP-only implementation. The real time aspects of the waveform made this software architecture unworkable for the GMR platform. To make matters worse, no Portability Toolkit assets beyond source code and limited documentation were available.

When ported to GMR, the waveform was re-architected to partition its real time components from the GPP to a DSP. This implementation ultimately succeeded in meeting performance specifications, but required device-specific timing management on the DSP to satisfy the waveform's stringent control timing requirements. As a result, the GMR SINCGARS porting effort had higher than anticipated porting costs.

Contrast this first port with a subsequent repartitioning of the waveform. For the first time, bit-true, fixed point models of the SINCGARS waveform real-time functions were created. The waveform was documented thoroughly, complete test vectors were created, and, most importantly, the waveform was repartitioned to eliminate the DSP device-specific code that was previously used to mitigate the DSP's nondeterministic timing. The two different architectures are illustrated in Figure 2. With a complete Portability Toolkit in place, as shown in Figure 4, the software architecture was modified to move the real time signal processing to an FPGA. In fact, the latest waveform port does not require the use of a DSP, consisting entirely of universally-portable, non-hardware specific GPP and FPGA code (making it ideal for platforms without a DSP, such as JTRS HMS). The resource utilization of the repartitioned SINCGARS modem in several different FPGAs is shown in Table 1.

Table 1: Utilization of L-3 Nova's JTRS SINCGARS modem core

Device	LEs/Slices		Mem Bits/RAMS		Multipliers/DSP48s	
	Number	%	Number	%	Number	%
Altera Stratix II EP2S60	7,431	12.3	58,742	2.3	25	17.4
Altera Cyclone II EP2C50	8,063	16.0	58,736	9.9	23	26.7
Xilinx Virtex-4 XC4VLX60	5,165	19.4	15	9.4	9	14.1
Xilinx Spartan-3 XC3S4000	5,174	18.7	15	15.6	9	9.4

Key improvements were made to the black side portion of the waveform where the modem resides. The software executes the same waveform modes and features and the signal processing techniques are identical. State machines were also added to improve the robustness of the waveform controller. The waveform functional partitioning was also designed to support porting spirals. For example, SINCGARS single channel modes can be demonstrated on a new platform very quickly as part of risk mitigation activities, while frequency hopping modes can be spiraled in later. In all cases, the SDR implementation was verified by demonstrating RF interoperability with legacy SINCGARS radios in all ported operating modes.

The net impact of the successively improved porting process was a SINCGARS waveform architecture, codebase and Portability Toolkit that provides an estimated 15x reduction of required waveform porting effort over the original waveform implementation. These waveform components will ultimately be portable to a stable SDR platform in fewer than 24 man-weeks.

5.2 Case Study 2: JTRS WNW OFDM

L-3 Nova licensed its WNW OFDM PHY to the JTRS GMR program and was responsible for integrating the waveform on the GMR platform. Since 2004, Nova has ported its OFDM PHY to 11 different platforms in seven different hardware configurations, as shown in Table 2. Our experiences porting the waveform led to the creation of the WNW OFDM Portability Toolkit. **The average time to complete a port and perform an RF demonstration is now typically 12 weeks**. Two realizations of the waveform partitioning are shown in Figure 3.

Table 2: Device configurations for the various WNW OFDM porting efforts

Platform Config	DSP	FPGA
1	TI C6416	Xilinx Virtex-II XC2V6000
2	TI C6416	Altera Stratix EP1S80
3	TI C6416	Xilinx Virtex-II XC2V3000
4	TI C6416	Altera Stratix II EP2S60
5	TI C6416	Altera Cyclone II EP2C70
6	TI C55x (OMAP)	Xilinx Virtex-4 XC4VLX60
7	TI C6416	Xilinx Virtex-4 XC4VLX60

In 10 of the 11 platform ports, the modem code did not require any modifications as it was ported to the target platform. In other words, the ideal definition of portability was attained in those porting efforts. In the remaining case, in which the code did require modification (Port 6), the software architecture was repartitioned to target a small form factor radio with limited DSP capabilities. Using the Portability Toolkit resources, the DSP signal processing blocks were retargeted to the FPGA. Since the Portability Toolkit provided the blueprint for the waveform, the recoding efforts and system validation costs were minimal.

7. CONCLUSION

The first iteration of waveform development efforts has fallen short of the community's expectations of waveform portability, and continues to pose a challenge to waveform porting schedules and costs. However, we've described recommended practices that, when coupled with the JTRS Portability Guidelines, have the promise to make true waveform portability achievable. Our recommendations for a waveform Portability Toolkit are supported with anecdotal evidence from multiple, successful ports of multiple waveforms.

8. REFERENCES

[1] "Restructured JTRS Program Reduces Risk, but Significant Challenges Remain", GAO Report to Congressional Committees, GAO-06-955, available from http://www.gao.gov/cgi-bin/getrpt?GAO-06-955.

[2] JTRS NED T&E Waveform Portability Guidelines v. 1.0, JTRS Network Enterprise Domain, April 13, 2007.