

PORTING LESSONS LEARNED FROM SOLDIER RADIO WAVEFORM (SRW)

Adam Blair, Tom Brown, Jonathan Cromwell, Sungill Kim, Ryan Milne

TrellisWare Technologies, Inc.
San Diego, CA.

ABSTRACT

The Soldier Radio Waveform (SRW) is one of the key waveforms driving JTRS and related Software Defined Radio (SDR) implementations, particularly for ground systems. Due to wide application space and the complexity of the highly parallel advanced PHY processing, SRW is one of the critical benchmarks in assessing the viability of a given SDR modem architecture to support JTRS requirements. TrellisWare Technologies, Inc. applied its commercial AID and PSP technologies to the digital processing for one of the primary modes of SRW and has ported variations of this technology to a half dozen different SDR platforms for as many different companies and applications. In the process a number of critical lessons about implementing and porting complex, parallel implementations of this nature to these types of platforms have been learned. Issues such as optimum partitioning between processor and Field Programmable Gate Arrays (FPGA), matching the technology to the architecture and porting from one chip vendor's family of products to another's are assessed. Projections on the future evolution of such algorithms and the technology on which it is instantiated are also included.

1. INTRODUCTION

TrellisWare Technologies, Inc. creates high performance iterative and trellis-based commercial modules for extremely high performance receive processing. These modules have been applied to support baseband Physical Layer (PHY) Processing for the SRW Combat Communications (CC) mode as well as other Likelihood based processing applications. TrellisWare's technology is derived from ground-breaking techniques initiated in academia by two of its co-founders and demonstrated in a research effort originally sponsored on an award winning SBIR for the Army Research Lab (ARL)[1][2]. Due to the highly parallel nature of the algorithms used to maximize performance, SRW is one of the more demanding waveforms being implemented on JTRS platforms and, as such, provides a good test of portability and processing considerations. TrellisWare has ported its commercial technology for SRW, as well as similar applications, to numerous different SDR architectures for several different

Prime Contractors. Lessons learned from these efforts are highlighted here.

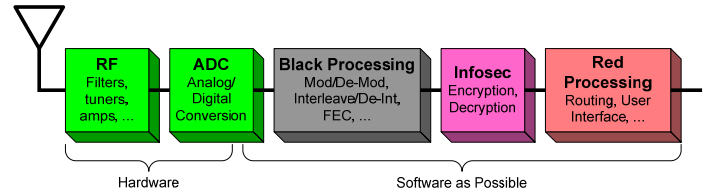


Figure 1: All Elements of a Waveform

TrellisWare uses an Adaptive Iterative Detection (AID) module to enable extremely high performance receiver implementation. However, this also creates significant complexity challenges for the receiver since the entire adaptive receive processing must now be repeated (“iterated”) several times, passing soft decision information, and still completing within the latency constraints required to achieve full real-time throughput.

A block diagram of the generic high level receive processing for a typical AID implementation, including the iterative segments, is shown in Figure 2. In order to combat severe multipath fading environments, the processing repeats the full receive baseband processing N times for N iterations, in addition to incorporating Per Survivor Processing (PSP), a highly parallel, trellis architecture that further impacts the partitioning and latency considerations.

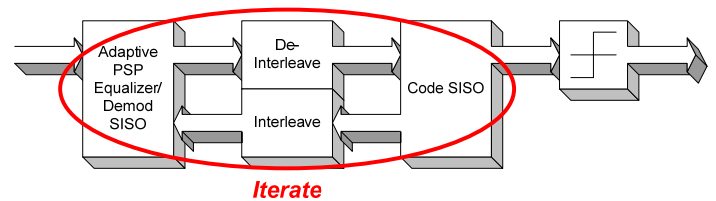


Figure 2: Receiver Block Diagram

The rest of the paper is organized as follows. In Section 2, the system considerations for porting are discussed. Section 3 discusses some of the tradeoffs between software and hardware realizations, while a few key do's and don'ts of design for portability are highlighted in Section 4.

2. SYSTEMS PORTING IMPLICATIONS

Even though TrellisWare's standard receiver core encompasses just the physical layer baseband processing, in order to successfully port the core it is necessary to carefully consider the whole system. Processing and hardware that surrounds the core will affect the performance and operation of the baseband processing done in the core. For example, the RF and IF characteristics, the networking protocol, and the power supply all will play a part on receiver core operation. Certainly, a well defined, comprehensive interface document and specification is crucial to make all the parts work together. However, not all operational circumstances can be predicted beforehand. A few examples of systems level lessons learned from porting and integration efforts by TrellisWare follow.

The iterative nature of the AID processing means that the various components of the core are constantly switching on and off potentially dramatically changing current draw. Legacy platforms often do not have power supplies that are designed for this. Consequently, the instantaneous current ramp-up caused by the iterative processing has been problematic for some platforms, even though the maximum current levels were specified as acceptable. This is an example of how introducing a new technology into an existing platform can lead to surprising new requirements.

Another issue that often arises concerning the front-end processing prior to the PHY core is that of automatic gain control (AGC). Two competing needs must be balanced. First of all, the initial (often analog) AGC must converge quickly so that samples into the receiver core are appropriately scaled and quantized. However, once the receiver core processing starts, the front-end AGC must be fixed or disabled. This is because the receiver core itself is adaptive, and having competing gain tracking loops is detrimental to performance. TrellisWare has often found that of the two issues the fast initial tracking is not a problem for legacy platforms, but the latter issue of fixing or stabilizing the AGC has to be introduced as a new requirement or feature for the platform.

Network layer considerations also play a considerable part in the porting process. Often times, performance in a new platform is not the same as in the original platform the receiver core was developed for. There can be gains in some areas, and losses in others. Also, often times a new platform implies a new network application, and so the requirements of the original platform might not be meaningful. Therefore, only a network level simulation can truly determine if any losses are problematic. It's our

experience that a network level simulation is invaluable in prioritizing which issues to attack first, and which to ignore. Without that kind of guidance, much engineering effort can be wasted trying to solve issues that have little impact on the overall network performance.

All of the above examples underscore how a successful port of the SDR receiver core is a total team effort. A successful port usually has a wide spectrum of engineering personnel represented – IF/RF, systems, board, FPGA, and network layer engineers. Also, establishing a core group of people that understand the waveform is invaluable when porting such a complex and cutting edge waveform.

3. HARDWARE VS. SOFTWARE

Hardware and software implementations both have certain arenas in which they excel. Software is ideally suited to more serialized structures with many conditional elements while hardware is at its best in highly parallel, pipelined applications. Today's system designer has a number of competing technologies to choose from for potential SDR applications (note that ASIC is included here as hardware accelerators are often an option, an approach almost always taken for commercial designs):

Table 1: Trade-Off in Processing Approach

Platform	Relative Computational Capacity	Relative Flexibility	Relative Cost
<i>ASIC</i>	Highest	Low	High (Low in volume)
<i>High End FPGA</i> (Virtex-5, Stratix-II)	High	Medium	High
<i>Low Density FPGA</i> (Spartan3A-DSP, Cyclone-III)	Medium	Medium	Medium
<i>DSP/GPP</i>	Low	Med/High	Low

While flexibility and cost are always important considerations, the computational capacity of the host platform is perhaps the most important consideration. If insufficient computational resources are available to support the required processing, nothing else really matters. Many modern waveforms and Forward Error Correction codes (FECs) utilize iterative techniques. These algorithms allow a direct trade to be made between algorithmic complexity and performance by adjusting the number of iterations. In addition for modern, trellis-based Likelihood techniques, the performance also increases

with the number of states in the trellis. However, the computational and routing requirements grow exponentially with the number of trellis states. Hence, for very complex waveforms where maximum performance is also desired an FPGA (or even ASIC) based solution may be a necessity.

It's important to note that Moore's law marches onward and all SDR platform's capabilities are constantly being improved. The TrellisWare SRW core originally required high end FPGAs, but as shown in Figure 3 below, this same processing is now able to fit into lower density FPGAs. This is true even though custom interface and control logic is almost always requested in the FPGA and TrellisWare's core comprises only a portion of the total FPGA processing. Fitting yesterday's complex waveform into tomorrow's powerful DSP/GPP is an exciting prospect. But equally exciting are the opportunities provided by tomorrow's FPGAs, where the extra processing power will be capable of supplying waveform performance considered technically infeasible today.

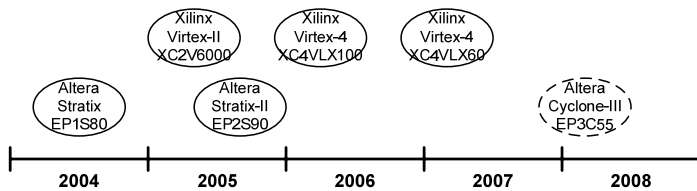


Figure 3: TrellisWare SRW FPGA Implementation History

For PHY layer processing such as SRW it is easy to take advantage of improvements that come with Moore's law. For example, as faster FPGAs become available the AID processing of the receiver core can be programmed increase processing iterations and metric bit widths, and thus increasing robustness.

High level, easy-to-read, portable code will drive both hardware and software implementations to their respective extremes. High level software will be extremely serialized and tend to be slow. For example, a 128-tap FIR is implemented as a *for-loop* with a single multiply-accumulate operation within it. Depending upon the capabilities of the compiler, this may or may not compile to utilize multiple parallel MAC blocks. Writing efficient and portable software for parallel hardware that is guaranteed to fully utilize a processor's capabilities is a challenge. Moore's law assists inefficient code by accelerating the processor clock speeds.

Creating high level FPGA cores for hardware will tend to result in the other extreme: excessively parallel, wasteful of space, and faster than needed. A fully parallel 128 tap FIR filter can be written in a few lines of VHDL code, but it may be 100x faster than required and fully occupy an FPGA. Writing a hybrid serial/parallel FIR filter that is as small as possible and only as fast as needed is a much more challenging task. Regardless of the choice of hardware or software, inefficient high-level code is rarely going to be an acceptable solution. The designer is almost always going to want to get the most out of his chosen platform by designing at a lower level.

One of the reasons for favoring software over hardware has traditionally been portability. The programmable logic vendors have worked hard in this area and a case can be made that programmable logic is more portable for waveforms than software. Software portability depends on many factors including design for portability and tool quality. Portability implies that only minimal changes will be required moving from one platform to another and therefore significantly reduced software risk. However, writing portable software is challenging and writing efficient and portable software is even more so.

While high-level C and C++ can be portable, the implementation can have very different performance on different processors. For example, a digital signal processor might prefer very simple loops to utilize the specialized indexing hardware while a general-purpose processor might prefer slightly more complex loops to take advantage of instruction and data cache. Future processor changes such as multithreaded or multiprocessor designs will likely have different optimizations

Programmable logic of an FPGA is best suited for parallel, pipe-lined logic processing found in trellis type architectures. While storing information required for real-time processing within the programmable logic is necessary, storing large amounts of non-real-time information is better done in a RAM or ROM. Implementing a complex state machine in low-level logic can also be challenging.

Moving initialization and configuration of a programmable logic module to software can be a positive trade. This allows the module to be simpler or more flexible. The processor can configure the module by setting parameters or filling in tables generated with arbitrarily complex algorithms in the software. For instance, storing interleaver tables for multiple waveforms within a programmable logic module (or even within a ROM) could be prohibitively large, while generating the table could be

a fairly simple software implementation. In this case, it is clearly beneficial to implement the initialization in software. In cases where multiple tables are required by a single waveform in real-time, a trade would have to be made between programmable logic storage and processing throughput requirements on software to reload the table as required. In this case, the additional design complexity may outweigh the resource costs.

Moving very complicated logic or state machines such as protocol processing to software can also be a positive trade. Implementing a particular functionality in software depends heavily on the processing required and the throughput between the software and programmable logic by the algorithm. However, there is a downside to splitting processing between software and programmable logic. There must be a well-defined interface between the two systems. This interface must be able to support not only the throughput requirements but also the latency constraints of the algorithm. It also must be portable in the sense that as the actual hardware changes, only minimal changes must be made to software.

In some cases, it has been a positive trade to include a very simple microprocessor within the programmable logic device to implement very simple control and sequencing of programmable logic blocks. This has worked well within some AID implementations to sequence the iterations. This has the benefit to being very tightly coupled to allow very direct control of the programmable logic modules with fine control over the resources available for data shuffling between modules.

4. DESIGN FOR PORTABILITY

Numerous high level lessons have been learned that can help in future designs for SDR portability. Flexibility in terms of both design and implementation is critical to achieve the high level of portability necessary to accommodate the diverse SDR platforms available in the military market today.

From a system level, it is necessary to have a design whereby efficient algorithmic approaches can be traded off with table-based implementations on significant sub-blocks. Inherently, algorithmic approaches are more logic intensive (be it MIPS in a DSP or LUTs in an FPGA), while table-based approaches are memory intensive. Since different platforms will have different amounts of processing and memory resources, it is necessary to have blocks of the system that can be moved from one type of resource to another.

The AID interleaver is an example of a sub-block that has been implemented with both a table-based look-up table (LUT) approach and an algorithmic approach. Initially designed as a table-based interleaver, the desire for additional modes and changes in hardware platforms drove this to the use of an algorithm. The algorithmic interleaver approach allows for efficient implementation without needing to compromise waveform performance. In this example, the algorithm was not initially designed into the system, and necessity drove the need to change the approach. While porting drove the necessity for the algorithmic approach, with hindsight, the algorithmic approach could have been used from the beginning of the design process.

If partitioning across multiple processor platforms is desired there are certain constraints to keep in mind. All internal modules should be designed to minimize the number of IO bits passing between them. And all inter-module communication should be designed to be transparently tolerant of increased latency. While minimizing the number of IO bits between modules is an important goal for maximizing porting flexibility, it cannot be at the expense of primary design requirements such as size and throughput.

Latency tolerant interfaces (such as hand-shakes) are not prohibitive, but they do sometimes come at the expense of additional complexity and small increase in design size. In general, the complexity can be overcome by good design reuse practices. Once a solid interface has been designed and tested, it can be reused in multiple modules and design size increases may be negligible as the interface logic is often very small compared to the data processing logic. Even with minimal-IO modules and latency tolerant interfaces, the designer must keep in mind that any design partition across FPGAs that introduces latency may have a significant impact on the overall core throughput. These effects may be magnified in iterative cores such as the AID cores. The FPGA partition points must be chosen carefully to minimize these effects.

Design for portability must also account for differences between different platform vendors and families of processors. For example, Altera and Xilinx take different approaches with memory resources in their FPGAs. Altera chose to have small, medium, and large memory blocks in their Stratix and Stratix II families. Xilinx provides one size in the Virtex families that fits somewhere between Altera's medium and large sized RAMs. Xilinx also provides the capability of using LUTs as distributed RAM.

Our experience has shown that designs typically have a wide variety of memory sizes. Shallow FIFOs are often used to store control information and as temporary storage in the main processing flow. Medium sized memory blocks are frequently used as ROMs to store tables for interleavers, or for storing pre-computed results for computationally intensive calculations. Large RAMs are useful for storing large amounts of data that would otherwise have to be re-computed with each iteration.

Considering both memory architectures from the start of a design allows for greater portability without having inefficient memory utilization. For instance, in the parallelization of an algorithm you may end up with a small RAM in each hardware slice that essentially performs the same function. Typically the memory can be architected in such a way that the small logical memories can be combined into one larger physical memory. This allows more efficient implementation in architectures with only one size of memory.

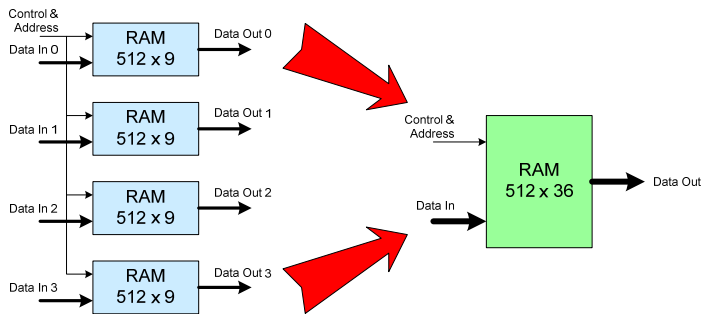


Figure 4: Combination of RAM blocks for efficiency in a particular architecture.

As designs are driven to small low-power architectures, memory resources are usually reduced dramatically. Therefore it is important to support alternatives, such as using off-chip memory or moving pre-computed tables into logic that directly performs the computation.

Other key FPGA lessons learned include:

- A single clock domain design greatly enhances portability and reduces integration time. For instance, various data rates can be generated in a transmitter core from a single input clock by throttling the data at the desired rate with an enable pulse. Running at a clock rate much higher than the data rate allows other regions of the core to be decoupled from the output data rate and therefore remain consistent as new modes and data rates are added in the future.
- When a core is developed independently of the surrounding logic, a well defined interface is critical to

a successful integration. This allows concurrent development of the core logic and the surrounding logic. When the final core is delivered, integration time can be greatly reduced if a test-bench and test vectors are provided along with the core. This provides a “golden” reference that can be used throughout the integration process.

- A portable design can not rely on device specific features as they may not be available on other FPGAs. For example, Altera PLLs and Xilinx DCMs offer the same basic features of clock multiplication/division and clock skew management. However each has its own set of advanced features that have unique control requirements. For portability, these features should be avoided unless they provide a compelling advantage.
- Embedded “hard” processors should also be avoided as the processor type is not standardized among FPGA vendors. A better solution is to use customizable “soft” processors that can be adapted to provide common bus and peripheral interfaces to the logic.
- One requirement for portable FPGA designs is avoiding instantiating FPGA primitives specific to a given family whenever possible. Most major synthesis tools provide some means of inferring a RAM, ROM, or multiplier from behavioral RTL. This allows a single codebase to support any FPGA family supported by the synthesis tool vendor, and allows simple migration to new FPGA families as they become available. Another advantage of using a third-party synthesis tools is that it ensures consistent interpretation and implementation of the logic across all FPGA families.

Most of these lessons can probably be summarized as trying to remain as flexible and generic as possible in the design and avoiding the temptation to make use of nice features on one platform that might not be available on another. Tools can help in this process but nothing can replace sound engineering and early portability planning.

5. CONCLUSIONS

Modern communications techniques, such as AID and PSP, offer tremendous performance gains but come at the cost of added baseband complexity in both iteration and parallel processing. Porting of these types of implementations is challenging, as TrellisWare has learned with SRW and similar implementations. Often programmable hardware is necessary for efficient

implementation, although there are certainly trades where doing part in software and part in hardware is helpful. Appropriately partitioning the design and staying away from features tied to a single family of chips while considering all of the system implications is essential. TrellisWare's experience with SRW proves, however, that with careful engineering and understanding of the approach, it is practical to port such designs within reasonable time and cost constraints.

6. REFERENCES

- [1] Thomas Carter, Sungill Kim, Mark Johnson, "High Throughput, Power And Spectrally Efficient Communications In Dynamic Multi-path Environments," *MILCOM 2003 Vol.1.*, pp. 61 – 66, Oct. 2003.
- [2] J.L. Cromwell, G. Paparisto, K.M. Chugg, "On the Design of a Robust High-Speed Frequency-Hopped Radio for Severe Battlefield Channels", *MILCOM 2002 Vol.2.*, pp. 900 – 904, Oct. 2002.
- [3] Adam Blair, Thomas Brown, Keith M. Chugg, M. Johnson, "Tactical Mobile Mesh Network System Design", *MILCOM 2007*