“ENGINEERING DESIGN STEPS FOR FAST, SIMPLE AND ECONOMICAL WAVEFORM DEVELOPMENT”

Wayne Bonser
Air Force Research Laboratory
Information Directorate
Rome, NY

ABSTRACT
The software defined radio (SDR) has the potential to drastically reduce the types of radios in the inventory since a single hardware package can be programmed to replicate the coding, modulation and other signaling features now used in legacy networks, plus adapt to next generation developments. Future Department of Defense (DoD) radios can benefit from this software programmability and thus simplify interoperability, maintenance, and life cycle cost concerns.

Methods for the rapid development and distribution of waveform software must be created in order to realize the full potential of a communications system that can change parameters such as waveform, operating frequency, protocols, and error correction. There needs to be a standard SDR waveform development process addressing all of the engineering design steps for faster, simpler and more economical waveform development.

This paper discusses the following issues and provides clear answers to these questions: Why do we want (or need) a Waveform Development Environment (WDE)? Can HW/SW decisions be pushed down to the lowest appropriate decomposition level? Should the language be standardized so that it has the widest potential applicability? Could the environment build upon the latest multimedia developments, so that they could be both more affordable and supportable? Should there be a definitive migration path towards future upgrades (e.g. Internet and web-based technologies)?

INTRODUCTION
Military communications systems have problems stemming from what had been a need for tailored and peculiar radio [primarily voice] connections driven by unique military missions to the current need for a seamless interoperable network of digital information systems employing multimedia capabilities. There are over 200 different radio systems in the military inventory and most of these are both non-interoperable and require individualized logistics support.

The current thinking is to employ gateways and software defined radios to help reverse the trend of non-interoperable and uniquely supportable stovepipes.

In order to achieve the full advantage of a software defined radio system, which can rapidly and easily change its features such as the modulation type, operating frequency, acquisition and network protocols and error-correction, there need to be methods to create, modify, test and distribute software expeditiously and inexpensively. A software defined radio waveform development process which addresses all the necessary engineering design steps for faster, simpler and more economical waveform development must be created. Likewise, such a development process will benefit from being standardized such that common tools are available that produce common interoperable software.

BACKGROUND INFORMATION [1,2]
Almost anything today needs to be examined and justified from a “business case” perspective: “What saves money for both manufacturers and consumers alike?” Many times what saves manufacturers and developers will also save the consumer during purchase. What saves the consumer money is often the ability to
competitively procure and cost-effectively maintain a system. Furthermore, as capability becomes more affordable, markets expand and may even mushroom creating viable market shares for both producers and third-party vendors selling component-ware [either to the producers or the consumers].

ENGINEERING STEPS AND CONSIDERATIONS

Why we want and need a Waveform Development Environment (WDE)
As outlined in an older DoD Acquisition Handbook (see Figure 1) the after-deployment logistics comprise the most significant share of the Life-Cycle-Cost (LCC) of the system. It is projected that roughly 70% of a system’s LCC is established by decisions made (or options ignored) during the initial phase of development when only 5% of the LCC has actually been expended (see Figure 2)!

For a MILCOM paper in 1998 [3], the authors used a Delphic survey of ten highly experienced professionals, from seven vendors and three government agencies. They requested estimates for Non-Recurring Engineering (NRE) costs in terms of time, staff and investments for SDRs. On the average, the survey showed that the experienced professionals believed that a software radio would result in 28-40% schedule savings and a 22-43% cost savings over conventional developments. Even more could be saved if there was an integrated tool suite to assist developers throughout the engineering phases. The lions share of development cost, many times, is created by trying to fix “problems” first noticed during integration and test (see Figure 3). If these problems had been caught earlier in the specification and design phases, the cost of fixing them could have been reduced to 1% of the cost that would have been incurred if the same problem was only discovered later, during the test phase. These statistics support the idea that those methods which enable early identification and resolution of problems and which enable flexibility in

Costly “Down-Stream Errors

"Fixing a problem in the requirements costs 1% as much as fixing the resulting [implementation].” IEEE Spectrum. August 1992

Figure 3

modification, upgrade, and enhancement of deployed systems will reduce the largest cost factors in BOTH the development and operation/support phases!

The features and capabilities of a software defined radio can be readily altered by adding or modifying software. Obvious features for such modification are modulation type, operating frequency, acquisition and network protocols, and error-correction. But to understand the
myriad of possibilities, we need to look a little deeper. Capability modifications can be internal or external. Adding new capability software can expand radio capabilities. Likewise, enabling different software already resident in the radio can alter configured capability. Control of the radio is a fertile area for software development. Let’s examine a few areas. Via a Human Machine Interface (HMI) an operator may want to “control” what the radio is connected to [both on the RF and I/O sides of the terminal], what band it operates in, what waveform it uses, whether it is used in plain-text or with some security, what is displayed, and what is audible. A network or communications manager may want to control or monitor RF filtering, power output, frequency-lockouts, antenna directivity and tuning, and RF sampling. A maintenance operator may want to control test software, maintenance logging, problem reporting, exercise BIT, employ loop-back testing, examine “test-points” etc. There are differing levels of control and perhaps different software [application] packages for use in a radio.

Waveforms themselves will need to be developed. Legacy waveforms currently make up the suite of required capabilities. However, as the means to employ and field waveforms becomes easier and more cost effective, the demand for refinements to existing capabilities may grow. If we look at the TADIL messages, the various upgrades to Have Quick, and the many evolutions of SINCGARS, it should become obvious that even legacy-systems [which were NOT inexpensive to change, modify or improve] have not been stagnant or immutable. Users may clamor for new system capabilities and new waveforms as we move further toward the goals of a digitized battlefield, a publish and subscribe-based information grid, and Vision 2025.

Since both producers and consumers benefit from having a “faster time to market” for new capabilities, any method to expedite the development and fielding process is important. Likewise the corresponding time-to-field or time-to-implement will be significant to the users – the faster, easier, and less costly a new capability the greater the demand is likely to be.

At the heart of expediting a “field-ready” capability is streamlining the development processes and catching most of the problems early-on when they are easier, cheaper, and FASTER to fix. Let’s decompose the process and then determine what could be streamlined and how. Before we begin the decomposition process we should remind ourselves that for a radio system there are actually two separate, although not totally distinct, designs and implementations. There is the development of the communications electronic “box” and the development of the “waveform,” which determines how information is prepared for use outside [RF, or Digital/Audio] the “box”. This is especially true when considering software defined radios, since the “Box” becomes generic and the “Waveform” specific. There are four basic phases in these overall development processes: Planning, Designing, Building, and the Integration and Test [of Hardware (Hw) and Software (Sw) – especially with SDRs where Hw and Sw may be developed independently].

The Planning phase includes: Requirements Analyses, Functional Analyses, the discovery of Derived Requirements and the documentation of these in a detailed System Requirements [and Performance] Specification. In this phase we are defining what the system ultimately needs to do to satisfy the goals and objectives for its intended use. An SDR, however, is much like a computer in that all possible “applications” are not fully known, so the development must be for a system that meets both known intentions and has the capability for growth and expandability to satisfy the yet to be determined objectives of its user. Simulations are helpful in determining if all the requirements have been identified and how prioritization, or weighting of requirements, may affect system performance.

The Design phase is where the system decomposition takes place. This is the process of asking, “How could this be done” for each of the higher level functions of the system in order to derive lower level functions. I use the word “could” since there might be a variety of acceptable ways to accomplish a function. Hardware and software partitioning and the resulting preliminary designs are created. At
times an arbitrary decision at this level creates problems down-stream. Many manufacturers simply follow the same design pattern they have used successfully before, and unwittingly arrive at sub-optimum designs [in performance, flexibility, or cost]. This is a place for Trade-off Analyses to be used, and where an iterative design, simulate, and analyze process can lead to better design solutions.

The Build phase is one where we “turn the crank” on our designs and see what we’ve built. In this prototype process, abstract designs on paper are transformed into physical form. Electronic circuits are built and software is coded. Typically developers use Computer Aided Design (CAD) tools and computer-assisted processes during a manufacturing phase. Once products [either a Hw or Sw component] have been prototyped, they are tested to see if they meet the requirements for which they were produced. Tweaks may be made, or new designs may be required, but eventually the component products are ready for integration.

The final phase is that of Integration and Testing. Here is where the “rubber meets the road.” Components are interconnected, hardware meets software, interfaces and interactions are tested, performance is measured, and any refinements are made. This phase unfortunately is where most of the problems are discovered and where fixes are most difficult to implement since previous design choices limit the resolution space available [otherwise incurring the costly redesign of larger functional components].

There are commercial tools to help in most phases of development. The DOORS database can be used to help in the requirements and planning phase, MATLAB and VHDL are useful for circuit design, UML can be used for software, and various tools for testing, such as LabView, COSSAP and SystemView are also available. However, there isn’t a fully integrated suite of tools, nor do the existing tools readily interface with each other. Because the SDR “box” is generic, an overarching and integrated CAD tool [i.e. a Development Environment] would enable the faster application of requirements to paper, built-in simulation capabilities, faster iterative requirement trades, direct to design capabilities (both for Hw and Sw), and both system-level and component level trade-off analyses; all without having to actually enter the build [or integrate and test] phase to find problems or optimize performance parameters. The option of employing modeling and simulation in early phases, and then using the model’s outputs to drive Hw/Sw allocation could save tremendous amounts of time.

---

**Decompose Radio - Waveform Design**

- **Menu driven**
- **Pull down list**
- **Simulation**
- **Automatically developed**
- **Detail Description document**
- **Computer Design**
- **Options – Alternatives**
- **Modeling & Simulation**
- **Boundary for Hw/Sw**
- **Computer Refined Design**
- **Hw & Sw detailed designs**

**Figure 4**

“Waveforms,” or system software applications, may be developed separately from the “box,” perhaps even beginning before the hardware is complete, or possibly after the system has been fielded, as in upgrades and enhancements. Therefore there would exist an independent “time to market” for software, and a similar need for a synergetic development environment to expedite waveform availability and help eliminate the more costly down-stream errors.

In the summer of 1997, in Real-time Engineering magazine [4], John Black discussed the massive budget cuts imposed on NASA’s space program, forcing the need to reduce development costs while being asked to accelerate development schedules. Three conclusions were reached: a) they needed to seek COTS real-time operating systems, which have less overhead and are sufficiently mature to have caught most “bugs;” b) they would need sophisticated development tools that would enable them to complete software within a tight budget and schedule; and c) that software development could not wait until hardware was available, the software would need to be
developed in parallel and then ported to run on the actual spacecraft hardware. They found that a good COTS OS with excellent development tools was able to allow 7 engineers to develop, debug, and port 150,000 lines of real-time application code within their 3-year time limit. Re-use of the code was planned for several more space missions. This article states: “With today’s short time-to-market and tight budgets, good development tools make the difference between success and failure.” They go on to say that 3rd party plug-in tools will keep manufacturers who facilitate these ahead of competitors who don’t.

If we can match development tools to the required engineering steps, we can see how an integrated tool suite could provide rapid development, enable quicker design optimization spirals, and save cost. Refer to Figure 4. The concept is to enable designers to create capabilities at the highest possible levels using pull-down menus for known parameters, and being able to draw upon comprehensive libraries of functional capabilities (which will later be implemented in Hw or Sw). Simulations allow immediate feedback on the adequacy of what has been chosen. The best of all worlds would be the eventual availability of an Executable Specification (see Figure 5).

Decisions on what is implemented in hardware and/or software.

A lead-in question should be “should decisions on Hw/Sw partitioning be pushed to lower levels of functional decomposition?” The answer lies in both performance and cost. Functional decomposition is a more complex issue for radio developers today than it was 30-40 years ago when radios were typically analog, single channel, single band and single waveform terminals; “tubes,” and the other analog components available, limited the design selection. Today we have at least three axes for decomposition: 1) Hardware – Firmware – Software; 2) RF – IF – I/O; and 3) Application-layer – Transport-layer – Link-layer [the ISO layers]. One could look at the radio as a cube devised on three axes. A developer’s eventual design appears as a fabric (surface), which weaves through this cube (see Figure 6). Assuming the appropriate modeling and simulations are accomplished and trade-off analyses applied, the best decomposition will enable developers to potentially re-use components (faster and cheaper development), optimize specific functions to meet constraints (i.e. real-time latencies, SWAP, etc.), and tweak performance to new levels of capability.

Open-system development practices eventually help answer the prime question “Can decisions be pushed to lower levels of functional decomposition.” In the DoD the term open systems has become a "buzzword", as fashionable to programs as it is to products. Unfortunately very little data has been made available from programs engaged in open system efforts and few have widely published their lessons learned. However, the DoD is committed to Open Systems and they have become a mandate for the development of electronic systems. For discussion in this paper I shall be using the US Navy’s adaptation of the
IEEE definition of Open System: "Systems that implement sufficient open specifications for interfaces, services and supporting formats to enable properly engineered components to be utilized across a wide range of components with minimal changes, to interoperate with other components on local and remote systems, and to interact with users in a style that facilitates user portability." Norman W. Kowalski in a paper [5] entitled: “Key Engineering Management Practices To Achieve An Open System In A DoD Environment,” explains, “Use of the term "sufficient" in the definition implies the performance characteristics of each individual system may be different and hence require different "profiles" of a particular system interface to be implemented, such as an office system and a missile interface.” He later amplifies the phrase “properly engineered components” saying that proper engineering of components involves controlling the open system interfaces such that both sides of the interface conform to specified standards. This involves identifying appropriate interfaces that meet both functional and performance requirements (profiling), identifying interface-conformant component implementations, and developing and maintaining conformant-applications (i.e. Hw & Sw components, modules, subsystems, etc.).

Take a look at any puzzle, the more complex it is, the more pieces that need to be included and assembled unambiguously. The same is true with the functional decomposition of a complex system. Sufficiency (profiles), consensus based standards, interfaces, and interface management (conformance), are key concepts of an open system. The objective of defining a “profile” is to ensure a complete and coherent subset of an open system environment within an application’s constraints and performance requirements. Such a profile does not limit any vendor’s components, implementations may differ, except at the interface level. In this definition a profile could be viewed as a very specific subset of standards to be applied within a system’s peculiar design space. Conformance to standard interfaces is not the end objective of an open system approach, realizing open system economic benefits is. By implementing upgrades and system component modification based on standard interfaces, being able to port or re-host applications in software to different vendors' platforms, enabling some level of plug-and-play of 3rd party component hardware, and, being able to readily accommodate these within an existing system, will grant both developers and users economic benefits.

Military objectives such as portability, interoperability, scalability, vendor independence [multi-source opportunities], and common training and logistics reflect various economic benefits associated with open systems. In order to enable flexibility [especially to allow developers to make wider and freer design and functional apportionment choices] at one level, there is a need to specify interfaces one-level-below the desired or expected implementation level. This way developers can “re-combine” lower-level functions as their intellectual property (IP) allows so they can attain economic or performance advantages.

The need for a standardized software development environment.

In November 2000, AFRL and the Software Defined Radio Forum (SDRF) co-hosted a Waveform Development Environment (WDE) Workshop. During a panel discussion at the end of the workshop a number of questions were posed that align with this article. At the conclusion of the panel discussion, consensus was evident on the need for and benefits of a “waveform” development language and a WDE. There was also general agreement that no integrated tool-set exists and that starting from scratch wasn’t the best choice either. A third issue dealt with a ‘standardized” Open-Source development model [a parallel perhaps to the LINUX environment]. Many thought the concept was good, but some offered an opinion that the economic benefit would be mainly for the user [i.e. government] vice the developer. Developers might be willing to give away tools “if” they perceive future opportunities for profitability; and customers might need to assume the entrepreneurial risks. However, incentives for innovators exist due to the dynamic spectrum opportunities. If developers
could have their IP ride upon an open-source model then they could make profits off their IP and have an added benefit of a tool that accelerates their time-to-market giving them an edge on their competitors. AFRL, DARPA, and DERA [UK] continue to work toward just such a capability. There are many lower-level tools that could be drawn upon. The WDE Workshop had participation from companies such as RACAL Ltd [now Thales Research], Foresight [formerly JRS, a developer of SVP and other tools], Elanix [producer of SystemView], Synopsys [producers of SystemC and CoCentric]. There is also university activity in this area, such as Berkeley [working POTOLEMY] and Vanderbilt [working Model-Integrated Computing].

*Building on the latest Multimedia developments and the migration path toward future upgrades.*

There are perceived benefits to developing an integrated system development environment leveraging web-based tools and processes. Open System, Web-enabled, COTS products could provide a solution but only time will tell. AFRL is pursuing WDE technology to enable interoperability of multinational wireless communications assets. Portability of radio waveform software onto independent radio hardware platforms is the goal of this program. AFRL’s approach to achieving waveform software transportability and enabling future upgrades, is the cooperative formulation of a Waveform Description Language to capture radio waveform functionality and a Waveform Development Environment to translate this description into operational radio waveform software. During this project, a WDL will be developed and used to capture two military waveforms. The waveforms will be described using the WDL, and then compiled, using mainly COTS WDE tools, into operational code for AFRL’s Software Radio Development System (SoRDS) hardware. An international interoperability test is planned with the United Kingdom, France and Germany.

**Summary**

Appropriate hardware and software decomposition and open system approaches will enable transportable application software. A Waveform Development Environment [tool suite] and a Waveform Definition Language are realistic and achievable capabilities to enable the full potential of software defined radios. Web-enabled, COTS based Open-Source products hold promise for making tool suites more effective and enhancements easier and faster to develop.

**References:**


