

# THE APPLICATION OF SOFTWARE-DEFINED RADIO TECHNOLOGY TO MULTI-STANDARD WAVEFORM GENERATION FOR TELEVISION AND RADIO

KEVIN BERNDSEN

Harris Corporation, Mason, OH

## I. SUMMARY

A new frontier of broadcast system design is upon us, brought about by technology advancement and adoption. Several approaches result in the ability to select from multiple waveforms for a single system. These approaches inevitably involve software and programmable processing to create each waveform for a specific air-interface standard. However, the combination of several technologies now allows the block diagram of a deployed system to be redefined.

A software-defined radio (SDR) platform includes a hardware composition and software framework which meets the requirements for flexibility through software. By combining the tools and components of digital signal processing with some of the best software architecture ideas from the computer and server domains, the result is a specialized fully capable development platform.

Basic software concepts which are familiar to computer users, such as applications, upgrades, and interfaces, also have well understood patterns and solutions for developers. Also, the interaction between the various pieces of hardware and software can be managed in a structured fashion. This allows developers to focus on maximizing the quality of the investment of the end user.

## II. INTRODUCTION

### A. Scope

There are several facets of the promises of SDR. One is the flexibility in frequency, bandwidth, and modulation type. These could be categorized as things which can change. Another facet is the way in which the software facilitates that flexibility. Much is said about the capabilities of existing platforms, but this paper intends to focus on the use of the underlying software technology in the design of a platform.

### B. Background

A radio system translates between an information format and a waveform defined by an air interface. Without making assumptions about the nature of the information or the air interface, a suitable radio prototype could not be designed.

Waveform generation is the process of converting an information stream into a waveform. This conversion is described by structural and algorithmic specifications captured by a waveform standard and any accompanying regulatory standards, such that the “air interface” is defined. For example, the ATSC A/53 standard [1] or the DVB-T standard [2] document their respective waveforms. Such standards are not lacking in specificity, given the complexity of the intended result and the potential for wide interpretation. A system complies with a particular standard if it performs the conversion according to the specification. Multi-standard waveform generation thus refers to the ability of a single system to satisfy multiple standards.

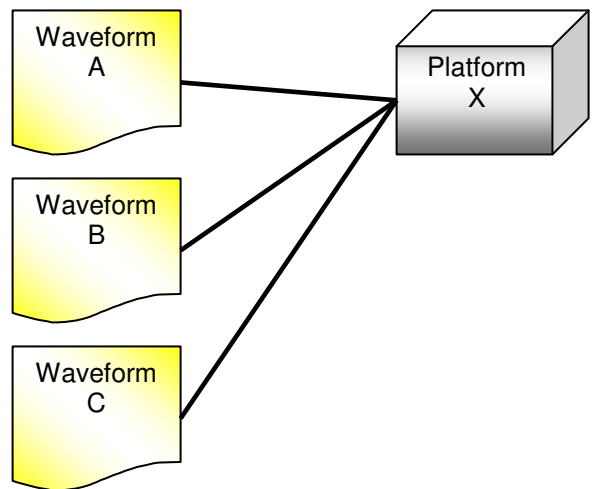


Figure 1: Multi-standard Platform

### C. History

Radio systems have a long legacy of purely analog circuitry and devices, but the design has changed from the inside outward. The starting point of the transition to digital implementations was not necessarily the advent of digital waveforms. Rather it was the capabilities of digital processing, which began to replace analog functional blocks. Digital processing has its share of issues and limitations as well.

Some now argue that the slate should be wiped clean and that patterns from the analog and digital legacy of radios should be avoided [3]. Others claim that the

problem of receiving and transmitting an analog waveform hasn't changed, just the implementation technology. Since analog circuitry necessarily remains a critical part of the system, designs must consider issues such as frequency selectivity and distortion. Modern systems are a mix of analog, digital, and software pieces, making the old analog radios seem like a more "pure" design.

Soon, "Software-based" won't be advertised anymore, just as has happened with "digital". Still, the majority of modern handheld devices have the technological maturity of PCs from the 1980's [4]. Only recently have platforms become more open for broad development.

The future of waveforms is divergent. The most proven techniques will be composed into configurable systems. These basic building blocks, such as baseband processing, downconversion, and synchronization will be used in intelligent systems to adaptively instantiate radio processing. The advanced techniques which push the envelope of efficiency and robustness will require highly optimized implementations on specialized processors. Hardware and software frameworks and tools will continue to be developed for research and eventually migrated to advanced platforms and products.

#### D. Software

Software, being a general term, needs some clarification. The software is the programs which run on a platform which ultimately runs on execution hardware. The software will only function on the platform for which it was made. However, the code base which produced the software is not restricted for a single platform because the same source code can be used by compiler tools to produce software for different platforms. This is true to varying degrees based on the type of platform and the characteristics of the source code. The issue of portability will be elaborated on in the design section.

The material input, output, and byproduct of software is data. It is the ability to create machine consumable information and thereby transform tasks into data processing which opens the door to employing software. Because digital signals are a form of data, the processing of those signals can be achieved with software. In turn, if the processing is software then the interaction in managing and monitoring that processing is handled more readily by software.

#### E. Software Defined Radio

In 1991, Joe Mitola coined the term "software radio" to "signal the shift from digital radio to multiband multimode software-defined radios" [5]. There are various definitions to delineate along the spectrum of software's role in the radio. At one end of this

spectrum is the term "Reconfigurable Radio" and at the other end is "Software Radio". Contemporary implementations are steadily advancing along this continuum, with an increase in versatility and abstraction facilitated by software. One definite partition of this continuum is the introduction of middleware. This additional complexity brings the advantage that the communication mechanisms are decoupled from the components.

An important distinction of SDR is the separation of waveforms from platforms and the process by which they are coupled. On one hand, the ability to apply a waveform to multiple platforms requires a platform-independent waveform representation. On the other hand, a platform is designed to host different types of waveform applications. These two characteristics are shown in Figure 2.

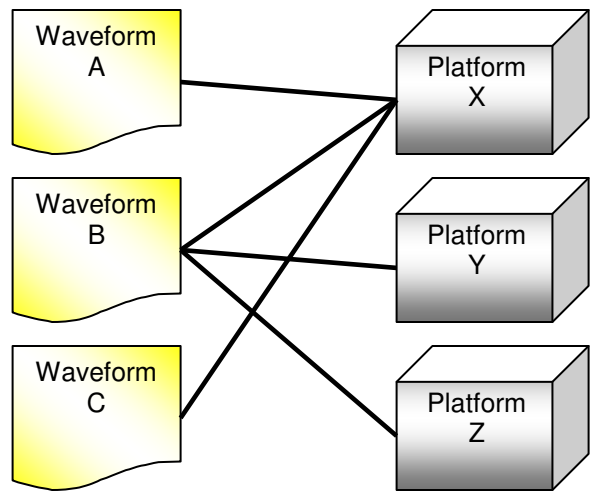


Figure 2: Waveform Portability

A waveform is "ported" to a platform by converting the waveform representation to use components which are compatible with or provided by the platform. The software architecture of an SDR is really just a unique combination of several proven methods and patterns of software in general. A few examples are: object-oriented design, object-oriented programming, layering, messaging, the model-driven architecture, and even automatic code generation. Each of these has their own technologies and tools.

The software infrastructure starts with the operating environment (OE). The OE is the equivalent of a computer's operating system. Elaborating on that comparison will establish a general taxonomy and provide helpful background information.

Many different operating systems exist, based on the type of computing hardware and the intended use. Hardware platforms may be categorized as server, desktop, portable, and embedded. The use of the OS

may be single or multi-user application hosting, multiple network-connected servers, or more specialized pre-defined single use applications. A few desktop, server, and real-time operating systems (RTOS) tend to dominate the share of their respective markets. Within these classes are other specifications based on scalability, processing architecture, or security.

An OE is indeed an extension of the general domain of an operating system. It includes generic services such as platform management, upgrade facilities, and connections to other networks or OEs.

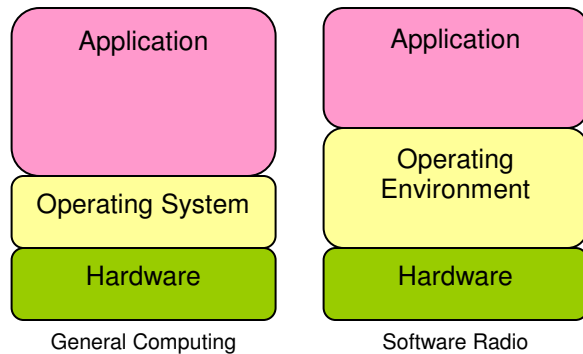


Figure 3: Comparison of Platform Architectures

Examples of existing OE specifications include:

- **OMG Software Radio** – a detailed specification intended to standardize the domain in the context of other Object Modeling Group (OMG) standards [7].
- **Software Communications Architecture (SCA)** – The JTRS sponsored standard. The SCA is the most widely used, specifically in military and defense radios. It is built upon other standards and protocols such as CORBA, IIOP, CCM, and POSIX.
- **GNU Radio** – A well-established open source project which implements several waveforms.

### III. DESIGN

#### A. Requirements

Platform design questions are similar for many domains such as vehicles, computers, and industrial equipment. Designing electronic equipment which will last for a long time is no trivial matter. The designer has to have a strong understanding of the important core or the “soul of the machine”. As it serves its purpose and things happen to it and around it, how is it expected to behave? Asking these questions up front leads to further questions whose answers lead to better solutions.

In order to produce a platform which is capable of serving the domain, many questions must be addressed:

How does one define what the platform is capable of? What are the primary and secondary constraints? How does one define the domain? Where are the tradeoffs of reliability, flexibility, limitations, cost, and complexity?

Although modulation and transceiver processing is similar for most waveforms, it cannot be assumed that a fixed parameterized processing block diagram will satisfy all applications. The resulting assumption is that the block diagram itself could change from one waveform to the next. By removing as many assumptions and constraints about a particular processing element’s place in the block diagram, then the element is only restricted in its relationship to other devices.

#### B. Principles

The advantages of quality design principles are difficult to appreciate. Often the promised benefits are not recognized because they simply achieve the desired result of making the design better. However, it is easier to recognize the lack of quality design principles when unsolved problems overburden the design. Experienced software developers at all levels can anticipate the solutions which must be incorporated into a domain problem. Design of a platform involves preserving the integrity of these solutions beyond the platform design and into applications.

Formalization is the act of establishing rules and adopting standards which will be followed by all parties involved in the platform. This enables a common starting point and language for capturing multiple perspectives. Modern software development utilizes practices which have been refined and accepted to varying degrees as best suited for broad circumstances. Instead of allowing each designer or developer to use their own discretion in adhering to these practices, formalization clearly establishes an agreement about these matters. While at first this may seem restrictive, in practice, as with agreement in other disciplines, it frees each individual from constantly questioning the advantages and applicability of each practice. An analogy may be the process of writing a report. While an individual may feel they have their ideas well understood “in my head”, it is the formalization of outlines, format, and grammar which allow the actual report to be produced “on paper”.

#### C. Problem

The general problem to be solved is to design and develop a complete platform which is capable of hosting multiple different waveforms. From a software point of view this can be restated as the ability to run different applications with the same specific hardware.

## 1. Portability

A common theme for SDR is the portability of waveforms across multiple platforms. The goal is to design a waveform and the associated models to allow for implementation on platforms with different architectures. This goal motivates the development and use of reference implementations, simulators, and distributed architectures which are less sensitive to the implementation platform.

The complementary perspective is true for the portability to a platform. A few possible approaches are:

- Anticipatory – determining the needs of the full domain of waveforms.
- Accommodating – allowing the platform to be modified to make a new waveform work.
- Restrictive – establishing barriers via criteria for implementation on the platform

While the third approach may seem detrimental to the platform, in practice this approach produces the most satisfactory outcomes for all stakeholders. This is only the first step in a broad problem of driving efforts toward increased portability, as in the figure below.

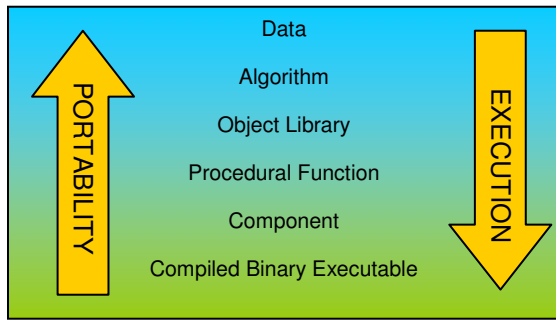


Figure 4: Software Portability

## 2. Data Management

Another aspect of the problem is data management. The ideal in portability produces more static and dynamic data artifacts. This originates from the characteristics of the platform and of each component. For every component there is associated data in the form of properties and models, whether it comes with the component implementation or it is gathered during integration.

## D. Characterization

Waveform components, from a black box perspective, have the following inputs and outputs:

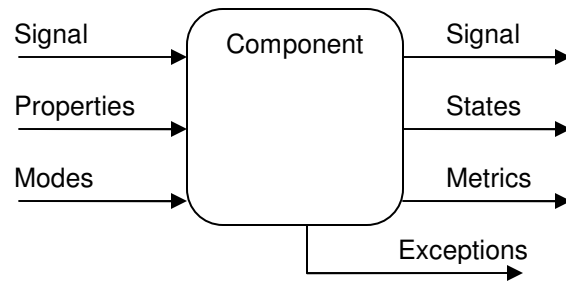


Figure 5: Component Characterization

Characterization in the software radio domain involves classification of logical devices, interconnections, physical elements, and signal properties.

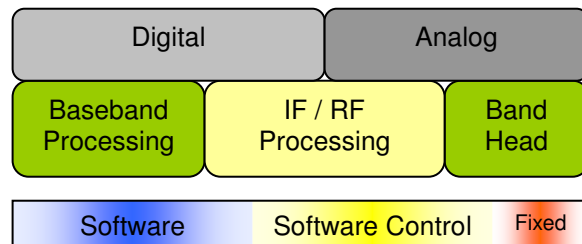


Figure 6: Basic Processing Chain

## E. Models

A homogenous reference implementation of the waveform processing is an executable model which can be used as a reference for verification of other implementations. Once the required functions are understood, an application model captures the integrated processing components. This platform-independent model (PIM) can be the centralized repository for all system information. The PIM along with a platform model can be combined to create a platform-specific model (PSM) which captures and documents the implementation.

## F. Communication

Communication within a system is a key architectural feature and it includes identity, notification, and messaging. In order for two components of software to interact, each component must have certain knowledge of the other. The simplest case of interaction is with a static object or function. For example, a function which adds two numbers may look like this:

```
double add(double x, double y);
```

The software calling this function supplies  $x$  and  $y$  and receives the answer.

However, software components are more complex than functions and require higher levels of communication. An example of a communication pattern would be a military officer commanding a soldier. With the

officer's authority over the soldier established, the officer must also have knowledge of the set of commands the soldier is capable of obeying. There is a set of commands which all soldiers understand and there are many additional specialized commands which specifically trained soldiers understand.

This simple example has several layers. Recognizing these layers opens the door to reducing rigidity and increasing flexibility in the architecture. From the previous example, once proper verbal communication is established between personnel, the next step is realizing that the communication does not have to take place "face to face". A commander can give orders over a radio (no pun intended) to soldiers or groups of soldiers who are geographically dispersed.

Had the communication been more primitive, such flexibility would not have been possible. Sophisticated software communication implies that messaging has formatting, patterns, and transport. In turn components acting as communication agents have ports to receive and send messages.

## **G. Platform**

The value of the platform is determined by its versatility and ease of use in implementing products. A platform may contain general purpose and specific processing elements. Regardless of the composition, the goal is to make the versatility usable. This is accomplished by:

1. Software components which are abstractions of hardware devices.
2. OE services which are used globally.
3. A fully configured OE for all operating systems used on all processors.

One problem which occurs is premature application of development effort to a specific final result. If a monolithic, single purpose application is developed for the platform, artificial constraints arise. Subsequent development for the next product invariably uses the existing implementation as a starting point. The problem with this approach is that information was lost and added in the translation from the original ideal processing to the actual implementation. A model-based implementation proposes to solve this problem by eliminating the sole dependence on existing implementations by new ones.

The most important principle in development of the platform is the appropriate efforts of the developers. They must have protection from premature constraints and sufficient incentive to produce versatile tools and thorough interfaces while avoiding assumptions about the final utilization. Inevitable discussions of imposing some constraints lead to well scrutinized decisions.

Formalizing the format of the work also delivers improved productivity.

## **IV. PLATFORM**

### **A. Architecture**

The architecture of a modern software-based platform is layered, similar to a general computing architecture (See Figure 3). In embedded approaches the complete application software is combined with the operating system whereas the software radio is similar to a server architecture in which the server applications are compiled separately and started by the operating system. This leads to the simple advantage that all sources need not be present when the platform software is compiled. This is a necessary condition if the development effort is partitioned in time or space.

Basic principles and patterns of object-oriented design are inherent in the framework. Examples include modular encapsulation, implementation hiding, adapters, and interfaces. Thus a specific waveform is implemented as one or more applications for the platform. The applications do the work of provisioning all of various components – configuring each one and "wiring" them together.

The building blocks of applications are components. The operating environment (OE) is the infrastructure into which components are deployed. The most important role of the OE is to facilitate communication among the components. The OE also provides basic services which the applications and all components can use. Typical services are communication, logging, event notification, fault reporting, and storage of persistent properties.

In implementation, the OE itself has limitations which require tuning such as partitioning across distributed processing resources. Each Executable Device contains a piece of software which facilitates its utilization within the OE.

The skills required to implement the "guts" of the components are different from those used to assemble and deploy a system. The framework provided by the object-oriented properties of the platform solidifies the proper boundaries of those skill sets and enables independence for developers. This also allows for interchangeable, functionally equivalent components.

### **B. Tools**

Platform compliance and interoperability is built into the modeling design tools themselves. When the system designer captures the design in the models, it is automatically prepared to be used by additional tools associated with the platform. This tool-chain allows changes to be made at the abstract model level. These

changes can become effective immediately without the need make corresponding changes at multiple abstraction layers.

### C. Proficiency

Making use of the capabilities of a complex platform requires proficiency. This refers to the understanding held by developers of the platform itself and of all the tools and forms of interaction. This quality can best be obtained by experience. An adult who is involved in training a child or a domestic animal achieves multiple results. The first result is the goal of a well-trained child or animal. The second result is the adult's intimate knowledge of the trainee which was acquired during the entire process.

The same is true of a platform – those who were most involved are most proficient. There is no excuse for developing a platform and not understanding its capabilities. But nevertheless this can happen, simply because without the proper structure, the complexity can easily become overwhelming. Proficiency can be transferred, but invariably experience is the best teacher. That experience can be used to continually make the platform easier to use.

## V. APPLICATIONS

### A. Diagnostic

One application is for a diagnostic program which is used to test system functionality. The application provides an interface to the low level configuration and parameters of all components of the system. It also queries and monitors these components during operation. What makes this possible is that the mechanisms for interacting with the components are predetermined and enforced by the operating environment. Components are designed such that their operation is not disturbed by the use of their advertised monitoring interfaces.

### B. Collaboration

In some cases, a waveform developer may desire to provide a waveform implementation or the proprietary core of a waveform. At the same time, they may wish to make their new waveform available on multiple platforms. This has been the case with iBiquity's IBOC waveform for terrestrial digital HDRadio® and Qualcomm's FLO waveform for mobile television. Since the implementation is specific to a particular processor, such as a DSP or FPGA, the integrator of the waveform must use the predetermined processor in their design.

Platform-independence would make the waveform more portable, but it would not directly result in an implementation. If a PIM was created using

standardized methods, the platform developer could provide models of the platform and tools to port the waveform.

## VI. CONCLUSION

The principles of software-defined radio bring as much advantage in development as they do for the end-user. Typically, developers who integrate individual waveforms with a platform are forced to work in multiple software layers simultaneously. Instead, by refocusing the effort which is applied in creating the platform, a framework of abstraction, tools, and patterns emerges. Thus, developers are less restricted in understanding the platform and they are free to apply their efforts toward optimizing signal quality performance and service mode flexibility, which benefit the end-user.

Reliable operation remains the highest requirement of an element in a broadcast chain. Secondary to that is flexibility. In the case of a multi-standard system, this depends on the likelihood and feasibility of changing the waveform. As adjacent technologies such as mobile handheld devices also become more flexible, there will be a need for upgrades of deployed air interfaces. Examples include ATSC to Mobile or even Analog TV to Digital TV. The design of the platform will determine the addition of features and the feasibility of new waveforms.

## VII. REFERENCES

- [1] A/53, Part 2: 2007. RF/Transmission System Characteristics. ATSC, 2007.
- [2] ETSI EN 300 744. Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television.
- [3] Farrell, Ronan. "Clean Slate Radio." SDR'08 Technical Conference Panel Discussion. 2008.
- [4] Viranen, Ari. "SDR Evolution in Commercial and Dual-Use Telecom Products." SDR'08 Technical Conference Keynote. Elektorbit Corporation, 2008.
- [5] Mitola, Joseph III. Software Radios. <<http://web.it.kth.se/~maguire/jmitola/>>
- [6] Bickle, Jerry. "Waveform Portability and Reuse Across Operating Environments: An Experience Report." PrismTech.
- [7] OMG. PIM and PSM for Software Radio Components V1.0. <<http://www.omg.org/spec/SDRP/1.0/>>
- [8] Waskiewicz, Fred. "OMG's MDA and Software Radio." <<http://vote.nist.gov/Berger1204OMGpresent.pdf>>. 2006.
- [9] Nelson, Rick. "Hardware and Software Approaches Implement Multiple Radios." *EDN*, Issue 26, 2008.