# RAPID PROTOTYPING FOR SCA

Mark Hermeling (Zeligsoft Inc; Gatineau, QC, Canada; mark@zeligsoft.com);
Justin Rice (Virginia Tech; Blacksburg, VA; ricej@vt.edu)

## ABSTRACT

Mathematical modeling tools for the signal processing chain are commonplace in the toolboxes of SDR (Software Defined Radio) developers. Typically, engineers model the physical layer of the waveform with these tools. Once the engineer is satisfied with the simulated behavior, the model is manually translated into high-level source code for DSP, FPGA, and GPP targets. The source code then is extended to behave as required with respect to the SCA standard.

Manual workflows are the norm in the development of SCA based Software Defined Radio. In this paper, we describe the work done by a consortium of BAE Systems, Virginia Tech University, The MathWorks, Xilinx, and Zeligsoft to look at automating this development process.

We look at rapid prototyping of waveforms through model-based design of both signal processing and component-based functionality. We discuss how the different models are related and how they can be used for automatic generation of required artifacts.

## 1. INTRODUCTION

Currently, SDR development is mostly a manual process. A team of engineers from very different backgrounds—such as signal processing, software engineering, and hardware engineering—labor together to develop the radio. Model-based design tools have been introduced in the different disciplines to assist these engineers with the tasks required to deliver the radio. However, these tools really only look at a single domain (signal processing, software engineering, or hardware engineering). Each tool allows engineers to model and validate their work, and generate artifacts, but these artifacts don't always easily combine with the work done in the other domains.

In this paper, we look at model-based design from a signal processing and SCA perspective. We investigate how these models can be used to rapidly prototype and build an SCA-compliant waveform (application) for a mixture of General Purpose Processors (GPP), Digital Signal Processors (DSP), and Field Programmable Gate Arrays (FPGA).

Traditionally, signal processing engineers have used tools such as The MathWorks Simulink [4] to model and simulate signal processing functionality. Both the sending and receiving parts of the waveform are modeled. Error patterns are injected so the performance of the waveform can be evaluated. Once engineers are satisfied that the waveform is sufficiently modeled, they manually translate the modeled behavior into source code.

This manual translation is a laborious process and a frequent source of errors. However, many mathematical modeling tools have built-in translation engines that can take the mathematical models and translate them straight into source code. This automatic translation increases efficiency and removes the manual translation step.

Additional modeling tools (for example, Zeligsoft's Component Enabler [2]) can model the SCA aspects of a waveform. These tools can translate the SCA aspects into artifacts—such as source code and XML descriptor files—required for SCA applications.

An integration between signal processing and SCA tools could combine the output of these two tools and directly generate SCA-compliant components with the full signal processing behavior. This would allow engineers to rapidly take the mathematical models, compile them into SCA-compliant components, and execute them on an SCA-compliant radio platform. However, at this time the integration between the signal processing model and SCA model is not available commercially.

A consortium of BAE Systems, Virginia Tech University, The MathWorks, and Zeligsoft has been investigating the integration described above. In this paper, we report on progress in this activity. We describe a process for developing mathematical models in The MathWorks' Simulink and then wrapping these models with source code generated from the Zeligsoft Component Enabler for execution in an SCA Core Framework (CF).

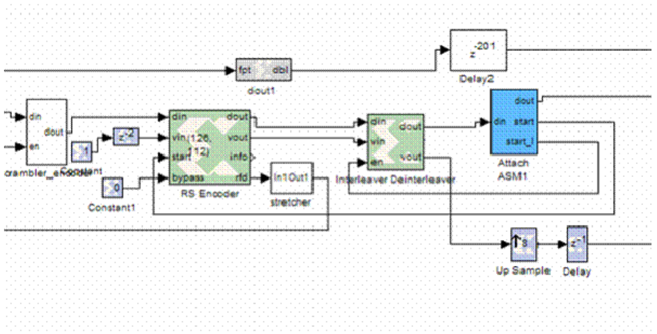
We also document the advantages and limitations of these tools and suggest future work to further accelerate SDR development.

## 2. BENEFITS OF MODEL-BASED DESIGN

The complexity of functionality in Software Defined Radios has been rising steadily. Engineers have been using mathematical modeling tools to manage this complexity. These modeling tools provide engineers with a way to visually describe the signal processing behavior of a waveform.

Modeling tools provide toolboxes containing default algorithms that can be used to rapidly create a waveform. Examples of this include different types of decoding or error correcting algorithms, source and sink for signals, error simulation, and so forth. Engineers can quickly model the desired behavior of a waveform with the standard toolboxes; they can also define their own algorithms and add them into the signal processing chain. Figure 1 contains a simple example of a signal processing waveform.

*Figure 1: Simple example of a signal processing model*



The processing chain can then be simulated and analyzed to see how much signal distortion would still allow uninterrupted data and voice transmissions. The end result is a complete description of the signal processing behavior of the radio. This is also referred to as 'the golden waveform.'

This method of designing signal processing behavior is referred to as model-based design. It is not new. Engineers have been using model-based design for years with great success. However, engineers would frequently model the waveform, and once the design has been sufficiently validated through simulation, manually implement the functionality on DSP, FPGA, and GPP processors. This manual step is not only laborious, but is also a major cause of how errors can slip into the implementation.

Modern model-based design tools now include an automatic translation from the graphical model to executable source code for a large class of DSP, FPGA, as well as GPP processor cores. This approach provides a number of direct benefits, including

- removing the possibility of error introduction
- significantly saving development time, since the manual step has been completely replaced by automation [1]
- removing the need for signal processing engineers to be intimately familiar with the development language for the processor

There are two more benefits that are less clear, yet definitely deserve some attention.

The first benefit is the option to perform hardware-in-the-loop simulation. Engineers can execute part of the behavior of the model on actual target hardware. This ensures that the automatically generated implementation performs as expected.

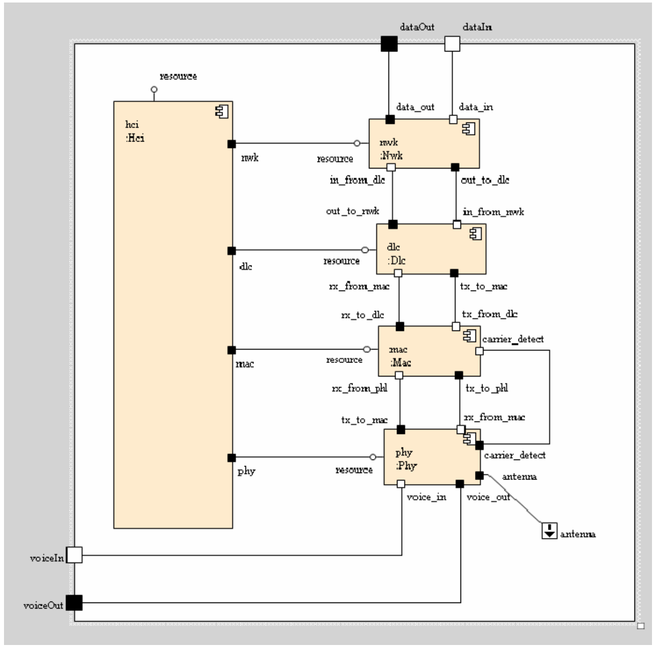
The second benefit is that the design model describes the algorithm completely independent of the processor that executes that algorithm. As such, it is an excellent way to ensure platform independence. The model can be translated to run on multiple platforms: GPP-, DSP-, or FPGA-based. This translation can often be completely or partially automated. The important implication of this realization is that model-based design provides platform independence at the model level, rather than at the source code level.

## 3. APPLICATION TO THE SCA

As stated in the previous section, model-based design has significant benefits in developing signal processing functionality. The question now is how this model-based design applies to applications that must adhere to the Software Communications Architecture (SCA).

The SCA treats applications as assembled and connected pieces of individually downloadable functionality. These pieces of functionality are referred to as components. A component can be executed on a GPP, DSP, or FPGA processor. An SCA-compliant platform consists of a number of processors; each processor can execute zero or more components. These components are then connected together through common communication buses such as CORBA, RapidIO, VME backplane connections, and so forth. SCA applications can also be graphically modeled, as shown in Figure 2.

*Figure 2: Component model of a sample waveform*



Typically the signal processing model accounts for a small part of the SCA mode—only the physical layer, as mentioned above. Nevertheless, this is a very important part of the software stack. Comparing the models in Figure 1 and Figure 2 begs the question as to how they are related. Figure 1 displays the signal processing content that comprises a part of the model in Figure 2.

Figure 1 displays the componentization of the software model. The visual representation defines how the components communicate together to deliver the full application (waveform) functionality. The application needs to be executed on a platform. This is graphically represented in Figure 3, where each component from Figure 2 is mapped to a processor (a device in SCA-language) on one of the boards of a Spectrum SDR-3000 [5]. Each of the devices in Figure 1 represents a GPP, DSP, or FPGA processor.

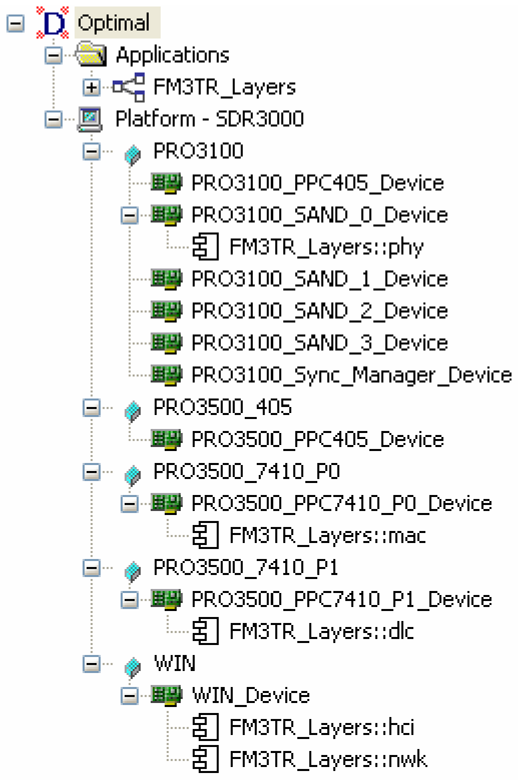
Each of the components in Figure 2 has related implementations that provide compiled binary code that implements the functionality of the component on a specific device (GPP, DSP, or FPGA).

The behavior modeled in the signal processing model (Figure 1) needs to be mapped to the component model in Figure 2. Typically, it maps the entire signal processing behavior into a single 'physical layer' component (Figure 2). However, this raises a few questions. Most notably, which device will the component be executed upon. Usually, this is a DSP or FPGA. Another important question is whether this device has the capacity to run all the signal processing content. If not, then we have to split Figure 1 into multiple components and map them to multiple devices.

This investigation becomes even more complicated if we want to execute the model on multiple, completely different platforms. The two step mapping of parts of Figure 1 to components in Figure 2 to devices in Figure 3 turns into a puzzle with multiple solutions. Each of the solutions to the puzzle will have certain performance characteristics, which need to be evaluated to ensure the implementation provides the performance required.

No tooling currently exists to facilitate this mapping (see also Section 5 on Further Tool Integration). The mapping is typically done by experienced engineers who have a complete understanding of the waveform requirements, as well as the platform offerings.

*Figure 3: Deploying the FM3TR waveform to the SDR-3000*



```
Optimal
  Applications
    FM3TR_Layers
  Platform - SDR3000
    PRO3100
      PRO3100_PPC405_Device
      PRO3100_SAND_0_Device
        FM3TR_Layers::phy
      PRO3100_SAND_1_Device
      PRO3100_SAND_2_Device
      PRO3100_SAND_3_Device
      PRO3100_Sync_Manager_Device
    PRO3500_405
      PRO3500_PPC405_Device
    PRO3500_7410_P0
      PRO3500_PPC7410_P0_Device
        FM3TR_Layers::mac
    PRO3500_7410_P1
      PRO3500_PPC7410_P1_Device
        FM3TR_Layers::dlc
    WIN
      WIN_Device
        FM3TR_Layers::hci
        FM3TR_Layers::nwk
```

## 4. COMPONENT WRAPPERS

Once all models have been sufficiently developed, it is time to build all the artifacts that the SCA requires for the waveform. The artifacts can be divided into two classes: XML descriptor files (the Domain Profile in the SCA lexicon) and the implementations for the components.

The XML descriptor files can be directly generated from the model in Figure 2. This is another instance of model-based design. Tools exist that can perform the translation of the graphical component model straight into the required XML descriptor files, without further human intervention.

Creating the implementation files for the components requires more effort. The signal processing behavior for the components is already given in Figure 1. However, the SCA requires that the components implement extra SCA-specific interfaces. These SCA-specific interfaces are required so that the SCA middleware (the Core Framework) can configure and control the components (through, for example, messages such as start and stop, as well as configuration parameters).

The model in Figure 2 contains sufficient information to generate the code that implements these SCA-specific interfaces. This code is often referred to as 'component code' or 'SCA wrapper code.' The only information that is missing is the connection between the behavior in Figure 1 and Figure 2. This is accomplished with a small amount of code, which merges these two models together.

The code required for the components depends on the device that the component executes on. The SCA standard is very clear on how to manage components on GPP processors, but it does not provide guidance for DSP and FPGA components. In this paper, we focus on how to integrate the two models for components executing on a GPP. Further work is required to experiment with generation for DSPs and FPGAs.

Components on a GPP use CORBA for communications between components. The component wrappers have to perform the following tasks:

- receive input data from CORBA buffers
- format this input and pass it to the functional block
- run the functional code to produce an output
- repackage the output into CORBA compatible packets
- translate SCA control messages such as start, run, and stop into control signals recognized by the signal processing code

Several simple models were implemented to test these ideas. The system model was developed using The MathWorks Simulink. The Real-Time Workshop package provides users with the ability to generate C/C++ source code from a Simulink model. There are a variety of target platforms to choose from; however, we only experimented with the GPP.

The SCA CF files were generated using Zeligsoft Component. Component code in C++ was generated for each component to provide an SCA-compliant CORBA interface. Short segments of wrapper code were then written and inserted into the component code. These wrappers receive data from CORBA, pass it to the functional model, run the model to generate an output, and then repackage the output to match the expected CORBA interface. They typically consist of several lines of simple and straightforward C code.

Several simple waveforms were generated for testing purposes. The OSSIE SCA CF [3], developed at Virginia Tech, was used to deploy these test waveforms on a GPP platform. Each component was able to properly deploy on the platform, process the input data, and communicate its results along the data path.

## 5. FURTHER TOOL INTEGRATION

Sections 3 and 4 describe in theory and in limited practice how to integrate the signal processing aspect and the SCA aspect of components. What is really needed, however, is a fully integrated tool chain.

Automating the details of the integration between tools is clearly on the list of further work; the following section highlights this in greater detail. However, there are still a number of questions to discuss.

As stated above, signal processing engineers work on the 'golden waveform,' the signal processing behavior of the waveform. Hardware engineers work on the details of the platform: the processors, the buses, and the AD and DA converters. Software engineers work on the hardware abstraction layer that allows the hardware to execute components (the 'BSP'). Software engineers also work on the higher level architecture, the components and their interconnections. The components cover both the 'golden waveform,' as well as higher order logic, which manages functionality like TCP/IP routing and so forth.

There is a mapping of pieces of the 'golden waveform' combined with higher order logic (1) to components (2) to hardware abstraction layer (3) to actual hardware (4). Each mapping is N-to-1 and has multiple possible solutions. Only a select few mappings actually lead to a correctly functioning waveform.

Now, how can we improve tooling to help engineers (be it signal processing, software, hardware, or systems engineers) design and evaluate all these mappings. The mappings cover multiple and very different domains—from mathematical signal processing, through software engineering to hardware design. Significantly more work is required to understand how the total end-to-end development process of signal processing, software, and hardware can be further integrated.

## 6. CONCLUSIONS AND FUTURE WORK

Modern model-based design tools provide an excellent approach for the rapid prototyping of SCA radios. The model-based design philosophy is a good match for the ideals of the SCA, including code reuse and platform independence. The combination of mathematical modeling tools, SCA interface tools, and automatic code generation allow system architects to quickly and easily progress from a system model to a running implementation.

The mixture of modeling tools is required since no one tool can provide a complete solution. Some hand coding is required to tie together the automatically generated functional blocks with the SCA interfaces. Automating the creation of these wrappers would provide a seamless transition from system model to implementation.

In this paper, we focused on solutions for GPP-based systems. Unfortunately, these platforms lack the processing power to manage waveforms that require high sampling rates. Specialized hardware, including DSPs and FPGAs, are typically called upon to provide the extra processing power in these situations. The SCA standards for implementing functionality on these devices, however, are not well-defined. In order for work to progress on developing wrapper structures for these platforms, the interfaces need to be standardized. With a common set of interfaces, wrappers could be generated for these specialized processing devices following the same principles as GPP-based design.

Vendors are already at work integrating signal processing and SCA tools. Solutions are expected to become available soon. The larger problem, as explained in Section 5, has not yet received significant attention. Building entire waveforms still requires significant manual development in a variety of separate, non-integrated tools.

## 7. REFERENCES

[1] Haessig, D., et al., “A Case-Study of the Xilinx System Generator Design Flow for Rapid Development of SDR Waveforms”, SDR Forum Technical Conference, Nov., 2005.

[2] Zeligsoft Component Enabler, http://www.zeligsoft.com/

[3] “Software Defined Radio (SDR) with OSSIE Open Source SCA.” <http://ossie.mprg.org/>.

[4] The MathWorks, Inc. http://www.mathworks.com/

[5] Spectrum Signal Processing SDR-3000 http://www.spectrumsignal.com/products/sdr/sdr_3000.asp

[6] Hermeling, M, “Component-based support for FPGA and DSP”, SDR Forum Technical Conference, Nov. 2006.