# BRIDGING DESIGN STAGES OF AN FPGA-BASED SYSTEM WITH A STRUCTURED ABSTRACTION METHODOLOGY

Karl Wagner (The MITRE Corporation, Bedford, MA, USA; kwagner@mitre.org)

## ABSTRACT

Modeling a complex system at increasing levels of abstraction reduces cost and schedule risks by providing early feedback on the effects of design decisions while also speeding up the overall design process, but it also introduces complexities to the design flow. Each level of abstraction requires a different skill set to design and analyze. Even the types of tools used vary between the abstraction levels. Transferring system requirements down through the levels and propagating results back up can be a complicated and error-prone process. Using a well-defined incremental approach with design artifacts that overlap the abstraction levels simplifies the process while still allowing designers to leverage the available features of their abstraction level. The MITRE Programmable Radio Technology (PRT) Laboratory demonstrated this approach through the implementation of a highly portable FPGA-based high bandwidth high throughput (HBHT) high data rate (HDR) modem.

## 1. INTRODUCTION

As FPGA-based systems become more complex, often spanning multiple FPGAs with varied interconnect fabrics, development methodologies must adapt to maintain productivity. For designs which must be portable to disparate hardware platforms, such as software-defined radio waveforms, the process is even more complicated. Furthermore, size, weight, and power (SWaP) constraints on systems require that designs be as efficient as possible. A robust verification environment, a modular design approach with a well-defined interface between processing components, and the ability to optimize the implementation are key enabling factors which mitigate the complexity of modern systems [1].

The MITRE PRT Laboratory is a multidiscipline electronic system rapid prototyping team focused on risk reduction and requirements specification with the mission of assisting the government in the successful acquisition of state-of-the-art communication and networking systems. The High Data Rate – Radio Frequency (HDR-RF) Test Waveform was created to ensure that modem hardware developed for the HDR-RF program has adequate computing resources to implement the proposed operational waveforms. The Test Waveform includes basic elements common to many HBHT waveforms including acquisition and tracking, modulation, filtering, strong forward error correction, and several selectable modes of operation. The complexity of the Test Waveform requires deployment on hardware platforms using several FPGAs. Portability and scalability were primary objectives in the development of the Test Waveform.

This paper describes the layered abstraction approach used in the design of the Test Waveform to enhance the design process and achieve the goals of portability and scalability.

## 2. MODULAR DESIGN AND ABSTRACTION

The development of a complex system can be simplified by breaking the system down into smaller self-contained subsystems. In this fashion the system can be isolated into blocks, each performing distinct aspects of the overall system. The blocks can be grouped into three categories: those which perform the algorithm, those which interface to the platform, and those which connect other blocks. The algorithm blocks are the most significant for the design of a portable system and are referred to as components in this paper. The link to the platform and connections between components are jointly described as infrastructure. The components and infrastructure rely on well-defined interface semantics so that they can be treated uniformly throughout the development process.

At each lower level of abstraction, the operation of the components and infrastructure is approximated in increasing detail. Modeling components at higher levels of abstraction, that is with less detail, is a common approach to simplify complex behaviors [2]. Several benefits are derived from the reduced detail including: different layers of a system such as hardware and software processes can be modeled uniformly; models run faster allowing more design iterations; and system-dependent decisions are delayed until late in the process improving the portability of the design.

One common approach used with high abstraction models is to directly convert the high-level description to a hardware implementation. Several commercial tools using a variety of description methods are based around such a transformation. When the overall project goal is to produce a prototype quickly, such methods can be highly effective. However for complex, robust and efficient designs required to be easily ported to alternate platforms, direct conversion

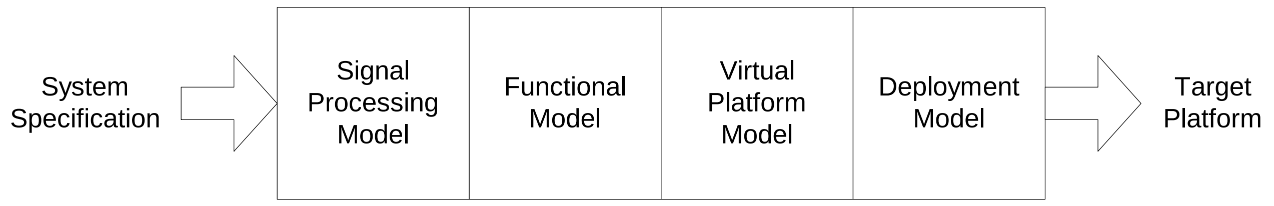

Figure 1: **Decreasing Levels of Abstraction**

has significant limitations. Lack of standardization between conversion tools locks a design into a specific tool chain and limits the flexibility of future implementations. High level languages such as C/C++ or Matlab/Simulink can offer a large pool of developers familiar with the language and well developed supporting environments. However, often only a specific subset of the high-level language is supported or a specific usage is required to generate the most efficient implementation. This limits the familiarity benefits and requires the high-level designer to understand the trade-offs made at lower levels.

Another approach to using high-level models directly is to build designs from a library of predefined components. While this approach can provide highly efficient solutions, only those options designed into the library are available. It is often cumbersome to develop new algorithms or alternate approaches. Library-based approaches are also affected by a lack of standardization, and connecting library components can require the development of nonstandard infrastructure.

For high-performance or highly constrained systems, being able to best leverage the features of the hardware can have a dramatic impact on the performance and SWaP characteristics of a design. While experienced FPGA designers are familiar with the strategies to optimize an implementation, system designers often are not.

## 3. CHOICE OF LEVELS

Various descriptions of abstraction levels have been proposed based on different views of the development process [2,3]. While any of these choices can benefit aspects of the process, the levels discussed here (see Figure 1) were chosen specifically for the design of portable and scalable signal processing applications based primarily in FPGA devices. The number of levels was kept small to limit the initial development effort required while still providing the artifacts necessary to port an application to dissimilar hardware platforms. This choice of levels and the terminology to describe them does not preclude other approaches as it should be possible to map alternate levels into the broad categories given here.

Each level encapsulates a particular set of skills and a distinct methodology used to design an application. By splitting the levels in this fashion, designers at a given level can concentrate on a specific aspect of the design and are not required to make cross-discipline decisions. The structured approach to the levels facilitates the communication of requirements and choices between the different designers. This division is conducive to designing a portable application since platform-specific decisions are delayed until the lower levels.

### 3.1. Signal Processing Model

At the top level, the signal processing aspects of the application are defined. The operations performed by the components are modeled while the infrastructure is completely abstracted. Although our development focused on FPGA-based processing, design at this level applies equally to other types of processing. Characteristics such as spectral containment, signal to noise ratio, and bit error rate (BER) are simulated and explored. Performance relative to a specification is visualized and evaluated at this level. The signal processing architecture including decomposition into discrete components is also defined.

Additionally, all design parameters are specified at the top level of abstraction. Although some parameters are not used at all levels, centralizing their configuration helps to streamline the verification process. This top-level configuration facilitates the scalability of the application and ensures consistency throughout the lower levels.

Tools used at this level should target these tasks, providing libraries of simulation and analysis utilities. A convenient way to chain components together allows the signal processing architecture to be explored and elements to be added to the chain as required. It should be possible to quickly change between algorithms and adjust their parameters so the impact of different options can be explored.

### 3.2. Functional Model

The functional model bridges the signal processing model to the virtual platform model. The distinction between components and infrastructure is more clearly defined and the components themselves may be modeled in greater detail. Thorough verification is required to ensure a robust implementation. This is particularly important when the

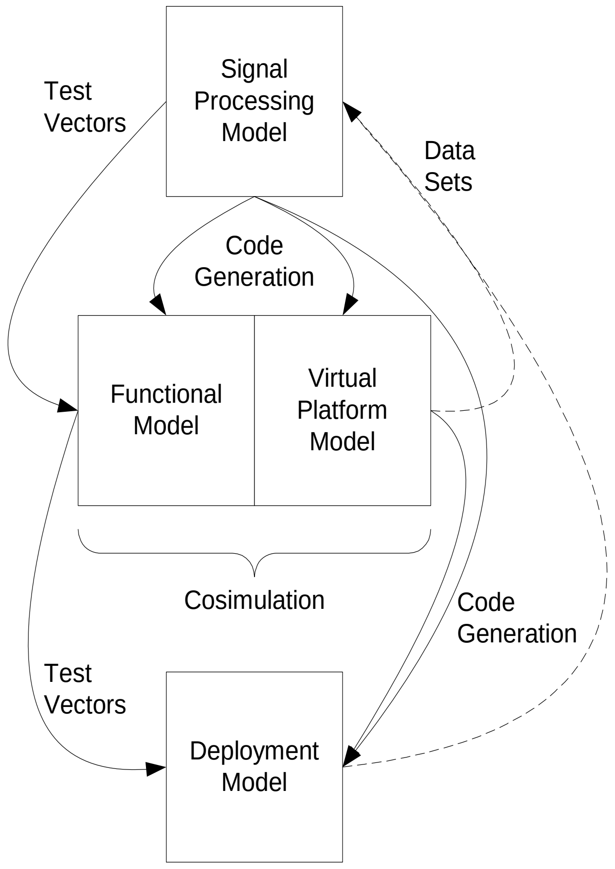

**Figure 2: Paths between Abstraction Levels**

context of the design may change in the future due to porting or reuse as with our Test Waveform. The degree to which the functional model must match the intended implementation depends on the desired robustness of the simulation. We chose to produce bit-accurate functional models so the implementation could be compared exactly with lower-level models. Even with this constraint, only the boundaries of the components must match exactly at each level. The internal algorithm can be structured as appropriate for the language used by their level. The infrastructure portion of the system remains highly abstract being represented as simple connections in the functional model.

The algorithms to be implemented and the connections between them have been defined, so the primary goal of the functional model is efficient simulation. The functional model must be able to process large data sets quickly with differing configurations and limited human interaction. Compiled software languages such as C/C++ are effective at this stage, have well established design practices, and can draw from enable a large pool of experienced designers.

The supporting environment for the functional model includes a collection of routines to stimulate and monitor the design. Since the simulations are intended to run primarily unattended, the environment does not focus on the visualization used in the signal processing model, but rather on collecting long-term metrics of performance.

### 3.3. Virtual Platform Model

The virtual platform model further refines the operation and structure of the components. It is based on synthesizable RTL code. This corresponds to traditional hardware design with regard to the components. Selecting RTL code restricts the implementation to FPGA- or ASIC-based processing resources, although the code still remains independent of any specific hardware architecture. The interfaces between components remain abstract, relying strictly on their prerequisite well-defined interface semantics; thus the models produced can easily be targeted to alternate platforms.

At this level, the hardware implementation of the algorithms is defined. The designer must have an understanding of hardware structures and how they can be arranged. Concepts such as pipelining and parallelism are important. The goals at this level include minimizing area, maximizing processing throughput, and reducing power consumption. Tools to perform netlist synthesis from languages such as VHDL or Verilog can effectively address these issues.

### 3.4. Deployment Model

The deployment model is ideally where the handoff from the application developer, who implements the components, to the porter, who must map the components to a specific hardware platform, occurs. When further porting is done, it can begin at this level as well, using the artifacts from the initial development for the previous levels. At this point in the process, specific characteristics of the target platform are introduced. The porter maps the virtual platform model onto the specific target platform resources. This includes mapping individual components and the connections between them to FPGA devices and physical links respectively. When the components have well-defined interfaces, it is not necessary to understand the function of the components; they can be treated as black boxes. Only the resources required to implement the boxes and the bandwidth and flow characteristics between the boxes are important. However, a detailed understanding of the target platform behavior is required. The abstract models of the infrastructure are replaced with models which accurately represent the behavior of the physical links. This might include writing synthesizable gaskets or wrappers to convert the well-defined interface used by the components to the particular protocol used by the target platform as well as using functional models of the physical hardware. A detailed simulation of the final system is produced by combining the platform-specific infrastructure models with

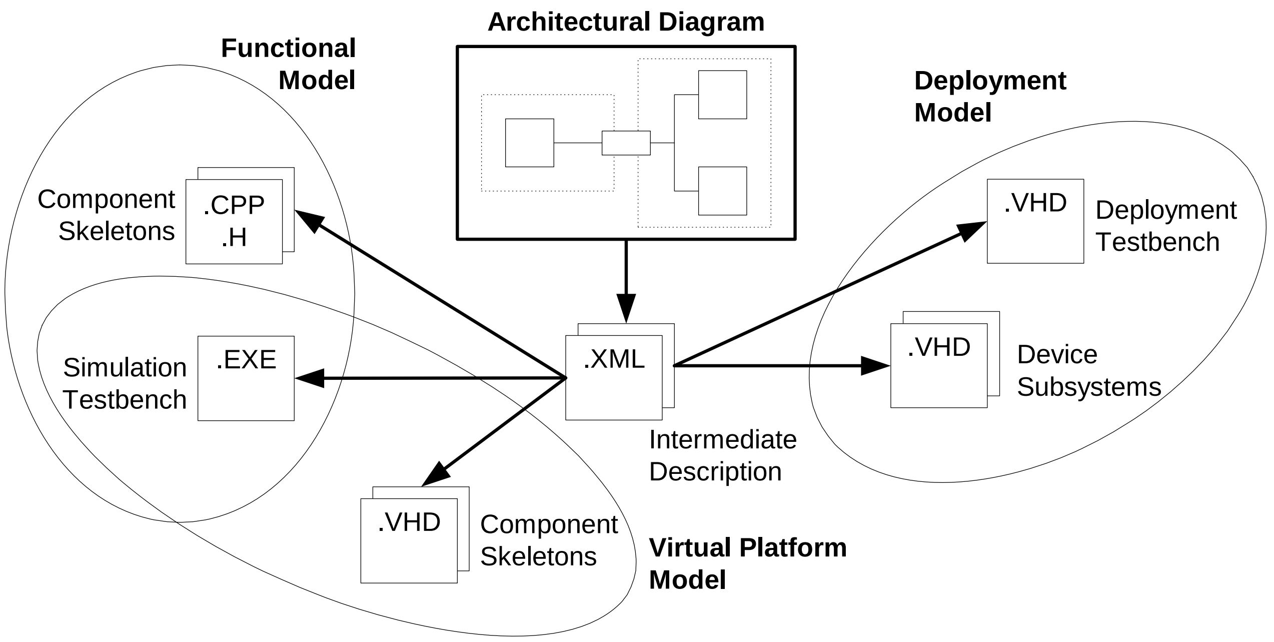

**Figure 3: Artifacts Derived From Architectural Drawing**

the component models developed at the virtual platform level.

The resource mapping step lends itself to graphical or heuristic-based tools working directly from a high-level block diagram of the application. Since this is the last stage before moving to the actual hardware platform, the simulation must accurately represent the timing characteristics of the system. RTL languages such as VHDL or Verilog are well suited to this task.

### 3.5. On-Platform Test

Ultimately, the application is more than just a series of simulations and must run on an actual hardware platform. Final integration is unchanged from other traditional approaches. However, if thorough verification has been performed at each of the proceeding levels, any inconsistencies in the design should have already been identified and corrected. This stage consists of loading the application onto the target platform and running it. It may also involve interfacing to analog portions of the design and testing interoperability with legacy systems.

## 4. MOVING BETWEEN LEVELS

The paths for communicating design decisions between the levels must be well-defined to reduce redundancy in the models while maintaining clean isolation. As illustrated in Figure 2, several paths were used in our design to move the information down the levels as well as feed results back up so they can be evaluated in the most convenient environment. Consistency is maintained by overlapping the artifacts from each level. The results of upper levels are used to validate the implementation of the lower levels, and, when necessary, the results from lower levels are fed back for analysis in the upper level.

### 4.1. Architectural Diagram

Since flexibility and scalability were primary design goals in our development, the components in our application contain a variety of configurable properties. The particular properties available vary by components, but two common examples are the bit width of the data path and the parallelism of the component algorithms. The bit width property is used to trade between resources required and signal processing performance. Increasing the bit width provides more precision, adding less noise to the calculations, but requires more resources to compute and required interconnection bandwidth. The parallelism property is used to trade between resources required and FPGA clocks rate used to achieve the required maximum throughput. Each of these properties is a good example of a design decision which spans abstraction levels. The benefit can most easily be evaluated at the top level while the cost is not seen until the lower levels.

All component properties and simulation settings are specified in the top-level architectural diagram. These settings are propagated through to the lower levels, which accept the configured values as illustrated in Figure 3. For our development, the architectural diagram was specified in the Simulink™ design tool from The MathWorks. The graphical interface allows easy visualization of the structure and a convenient method to navigate the various settings.

We use an XML-based intermediate structure to capture the information for parsing by scripts used in the lower levels.

The graphical interface of the architectural diagram is leveraged further to allocate components to processing elements during deployment. The components in a flattened diagram can be grouped to indicate device subsystem boundaries. A porter can take the same flattened diagram and quickly choose a different grouping appropriate for their target platform.

### 4.2. Code Generation

Implementing the algorithmic code separately at each level of abstraction using the language and architecture appropriate for that level can give further insight into the algorithm. It can also provide verification that the specification for the algorithm is correctly interpreted. However, implementing the structural code for each level can be tedious and error prone and provides no additional insight. Working from the architectural diagram, skeletons containing the structural code for each component are generated. By parameterizing the models, the configuration information defined at the top level is passed down allowing the design decisions to be incorporated and designers to focus on the tasks their level is suited to handle.

The component skeletons and supporting code for the functional model is generated from the XML representation of the architectural diagram. This includes all of the interface code to customize and connect each component to the simulation infrastructure. The component developer writes the underlying parameterized algorithm model as a simple C++ object. If a porter chooses to modify configurable parameter settings to fit their target platform, the code generation updates the simulation without modification to the algorithm model. As with the functional model, the XML representation is used to generate component skeletons for the virtual platform model. The developer fills in the algorithm using standard HDL code. The porter can later adjust parameters from the top level without touching the underlying code. Finally, at the deployment level, structural RTL code including the required internal infrastructure code is generated based on the device boundaries selected in the architectural diagram. The developer or porter is presented with a single aggregate component subsystem which he connects to his platform-specific infrastructure to complete the design.

The generation of component skeletons and supporting code removes the tedium of generating the connecting structural code but leaves the developers free to leverage the full language used for each level in the development of the algorithm. Code generation also allows all configuration information to be quickly fed from the top-level description without requiring the individual model levels to be modified directly.

### 4.3. Simulation

Verification is performed by the developer to ensure that the design operates as expected. Similarly, the porter must verify the deployment model to ensure the platform-specific infrastructure does not impact the behavior. Another important aspect of verification for the porter is to ensure confidence that the design operates as advertised. Being able to quickly generate and run a simulation helps satisfy these simulation goals.

The primary algorithm verification is done between the functional and virtual platform models. The infrastructure for the functional model is written using SystemC, a system-level language layered on top of C/C++ [4]. SystemC provides a convenient mechanism to directly compare the functional and virtual platform models. The functional models written in C/C++ can be simultaneously simulated with the RTL virtual platform models using commercial co-simulation tools. The user can enable co-simulation on a per-component basis from the architectural diagram to optimize a simulation for coverage or speed. Scripts allow the simulation to be configured, built, and executed with a single command to rapidly run through various scenarios.

The signal processing model and deployment model are linked to the functional/virtual platform models using more traditional test vector sets. Sample data sets and their expected results are generated by running a simulation at the higher level. The input data is then used to stimulate the lower-level simulation and the output is verified against the expected results. Using simple test vectors provides the highest degree of compatibility among tool sets used by potential porters. Although fixed vectors can limit test coverage, being able to quickly generate new sets for different configuration helps mitigate this shortcoming.

We have designed our infrastructure to facilitate use of test vector files by selectively overriding each component’s input and capturing or comparing each component’s output. As with other configuration settings, these selections are made from the architectural diagram and passed to the simulation via generated code. Parsing the captured data in the signal processing model also allows visualization of the results from other levels.

### 4.4. Actual Performance Evaluation

When a new target platform is more resource-constrained than the original platform, it may not be possible to match the performance of the original design, but it is still useful to port the design with reduced performance for partial compatibility. Using the top-down, layered approach, alternate settings can quickly be selected and fed through to low-level simulations. The captured results can then be inserted into the top-level model to characterize the impact of the changes and evaluated against modified system constraints.

Low-level model feedback is also used when the system or functional models do not provide a bit accurate representation of the RTL algorithm. While we chose to use bit-accurate models throughout the process, it may be easier to quickly generate higher-level models which merely approximate the end results. The simulation feedback is used to characterize the actual performance compared to the initial approximation. The simulation environment we developed can still be used to verify the design by switching from an exact comparison to a windowed one. Although co-simulation will be less thorough since small inconsistencies may pass undetected and the run time of each configuration permutation increases since the results must be passed back to the top abstraction level for analysis, this approach may be reasonable for some designs.

## 5. FUTURE WORK

One of the criticisms of existing commercial tools is the lack of standardization which limits flexibility when porting a design. While the process and artifacts used here have greater flexibility of tool choice and target platform, they still rely on a customized environment. Several standards exist which address different aspects of the design process. For example, specifications from the Object Management Group (OMG) [5] present a way to describe the architecture of an SDR application. The IP-XACT specification from the SPIRIT Consortium [6] addresses representing and passing configuration information between different tools. Adapting our environment to leverage these standards would extend the flexibility of our approach. Also, as commercial tools for various aspects of the approach converge on standards, they could be integrated into the overall flow.

Raising the simulation integration to include both hardware and software aspects of a system also provides an interesting possibility. Since the functional model is written in C/C++, it seems natural to insert algorithm code targeted for general purpose processor (GPP) resources into the simulation. Additionally, code to control the system and implement higher-level protocols which typically is implemented on GPP resources could be integrated with the FPGA simulation to produce a more complete model of the system.

## 6. CONCLUSIONS

By separating the development into distinct abstraction levels based on the tools used, developers can focus their efforts on the aspects of system design with which they are most familiar. System designers are not required to understand details of RTL design and the RTL designers still have the freedom to leverage the full range of implementation choices. Having a well-defined, structured approach allows intent and impact to be communicated between levels of abstraction. These techniques are particularly suited to designs which are intended to be portable or reused in the future, since the platform-specific portions are isolated to the final level.

The supporting environment of scripts and utilities we have developed facilitates the structured approach and simplifies steps where lower abstractions levels can derive information directly from higher levels. Our implementation of the HDR-RF Test Waveform helped refine the approach and demonstrates its value.

## 7. REFERENCES

[1] K. Skey, J. Bradley, and K. Wagner, “A Reuse Approach for FPGA-Based SDR Waveforms,” Milcom 2006, October, 2006.

[2] D.C.Black amd J. Donovan, *SystemC: From The Ground Up*, Springer, 2004.

[3] T. Kogel, A. Naverinen, and J. Aldis, “OCP TLM for Architectural Modeling”, July, 2005.

[4] IEEE Standard SystemC Language Reference Manual, IEEE Standard 1666-2005, 2006

[5] The Object Management Group, “PIM and PSM for Software Radio Components Specification”, Version 1.0, March, 2007.

[6] The SPIRIT Consortium, “IP-XACT v1.4: A Specification for XML Meta-data and Tool Interfaces”, March, 2008.