

# Application of Foresight's Technology to Implement a Waveform Development Environment

Bradley Logan, Deborah Wilson  
Foresight Systems, Inc.  
Anaheim, California

## ABSTRACT

*Communications are crucial to all aspects of life, whether they be individual cell phones, battlefield communications, or public safety emergency workers.*

*Because our globe is shrinking, it is imperative that radios communicate regardless of the originating countries, agencies, and vendors. To achieve interconnection of disparate waveforms and radio architectures, a mechanism must be available to efficiently develop and test waveforms, to prototype new Software Definable Radio (SDR) architectures, and to download waveforms in the field. At present, such SDR applications are developed using disparate traditional SW and HW tools that do not support a HW/SW co-design process for a total system synthesis.*

*Foresight Systems, Inc. has developed several tools that address a methodology for this waveform development environment, how to model functions and architectures for the environment, mapping functions to architectures, simulation and performance calculations, and automatic code generation for systems in an integrated development environment (IDE).*

*In the SDR field, Waveform and Control domain libraries must exist as well as architecture libraries with a rich variety of multi-processor models, operating systems (OS), protocols, and I/O drivers. Such libraries are necessary in order for the designer to perform HW/SW co-design and system trade-offs to construct various instantiations of communication waveforms dynamically downloadable to a software mobile radio.*

*The application of Foresight Systems' technology to the development of a Waveform Development Environment for Software Definable Radios is currently under active discussion between Foresight Systems and commercial firms.*

## INTRODUCTION

Foresight has developed a technology product, referred to as ASDEN (Advanced Systems Design Environment, a methodology and framework for integrating systems design tools) that can be applied to the domain of waveform development. This paper discusses the issues involved in building this "Waveform Development Environment" capability via ASDEN. This paper explains how an enhanced ASDEN can:

1. Easily represent the functionality and run-time environment of an application (e.g., communication waveform) as a block diagram and test/simulate its functionality.
2. Map that application onto a heterogeneous multi-processor platform.
3. Analyze candidate designs and perform hardware/software tradeoffs between mixtures of general-purpose

microprocessors (MPUs), digital signal processors (DSPs), and custom circuits represented in field programmable gate arrays (FPGAs) or ASICs.

4. Generate software load modules for the system, including schedulers, operating system, interrupt handlers, I/O drivers, and initialization code.
5. Symbolically debug the system on the target hardware platform using unobtrusive probes and a user-friendly graphical diagram of the application (preferably the same diagram developed in step [1] above).

These concepts already exist generically in the ASDEN methodology. Foresight Systems believes that ASDEN would be ideally suited for accommodation and implementation of the functionality needed for waveform representation and related compilation onto various hardware architectures. We will address this strategy in later sections as it relates directly to the waveform development capability. However, the reader will benefit from an orientation about the features and general methodology of ASDEN, as covered in the following two sections.

## Summary Features of the ASDEN Methodology

The Advanced Systems Design ENvironment (ASDEN) provides a design flow methodology for the user to design an application system from the top down. System-level requirements are specified and captured with the aid of tracking tools that create databases of application requirements. The functional requirements are then represented in the form of a high-level executable model using a block diagram paradigm. A set of candidate target architecture models are separately specified and captured. The tool set leads the user through HW/SW partitioning of the application to the candidate architectures, and provides analysis of the system performance. With a functionally correct, HW/SW Co-Designed application, the framework then automatically generates code according to the partitioning. This code generation produces load modules ready for the target platform integration and validation.

The methodology is driven by high-level functional requirements, and is able to analyze alternative ways to implement the captured application design requirements. Such an approach requires the support of libraries of building blocks both for functional and implementation-related tasks. In this context, "functional" means data processing or the software to do so, and "implementation" means the hardware architecture and components on which the software executes. To implement such a design system, the following features are required:

- A rich library of signal processing functional blocks, or primitives, and the ability to create and validate new primitives.
- A platform model concept incorporating systems components, connection rules, and the ability to create and validate new architecture instantiations.
- A populated library of reusable blocks that characterizes attributes such as timing, code size, and power dissipation for implementation of the functional blocks on potential target platform components.
- A high-level API (Application Programming Interface) that allows the designer to functionally represent low-level hardware concepts in the application domain language (e.g., RF and Waveforms) and corresponding implementation of the API elements as blocks for functional diagram use.
- Several operating systems from which to choose, with each of them characterized similarly to the library of reusable functional blocks.
- One or more compilers for each processor in the design, where the quality of the code generated by the compilers has been incorporated into the libraries of block implementations for each target processor.
- The ability to generate schedulers that preserve the data dependencies, sample rates, and event handling inherent in the application and that are optimized for target, embedded heterogeneous networks.

### **Overview of ASDEN Methodology**

A complete methodology is required to achieve an optimized, cost-effective systems design, independent of any specific tools or functional blocks. The methodology does not dictate the use of specific tools, but it requires that certain kinds of tools be used. If unavailable, then the needed tools must be developed. After the following methodology overview, we present a summary of the tool configuration used in the ASDEN prototype. The following discussion outlines the design steps illustrated in [Figure 1](#) which shows a Waveform Systems Engineer building an application with ASDEN.

#### Application Requirements Capture

Before beginning a project, the competent designer ideally strives to state goals or requirements as unambiguously as possible, and to validate them for consistency and completeness. ASDEN delivers functions that support this approach by delineating the design issues as well as the performance characteristics of the application, and by aiding in the traceability of requirements to design and implementation. High level models of functional and architectural designs are related directly back to requirements via tools. Automation thus provides productivity tools to help the designer by providing a structured approach to capture and validate the requirements, which will be met by the design and implementation.

Another long-term benefit of using an automated approach is the capability to state requirements in a format for automatic extraction and application to the design process. For example, in

terms of waveform applications, such requirements might include operating frequencies, supported data rates, and electrical power budgets available for a candidate design. Making the requirements data available to design tools integrated into the methodology allows such data to drive the design actions of the tools.

#### Application Model

The application model refers to a functional model of the application that captures the data processing and control expressed in the requirements. Such a model also acts as a living executable specification of the requirements as intended by the designer.

#### *Graphical Modeling with Block Libraries*

The preferred method of capturing the model is with graphical tools using a drag and drop drawing paradigm for construction. This approach depends upon libraries of reusable functional parts or blocks from which to build an application. Each of the blocks is built with reuse in mind and has parameters that control the behavior of a block instance. A simple example would be a multiplicative gain block where the parameter is the gain factor.

The graphical approach allows the models to be self-documenting in that the model is a hierarchy of pictures or graphs depicting the processing, data, and control flow elements of the application. These graphs are made up of standard blocks that perform well-defined functionality and are familiar to designers. This facilitates communication among a design team and allows each designer to concentrate on the overall processing and structure of the application rather than on the details of coding.

#### *Custom Algorithm Development*

The library mechanism is user-extensible in order to accommodate the addition of new algorithms by designers as they try to meet the unique needs of an application. Whether for standard or customized development, the system separately validates the functionality of each new library block, characterizes it in terms of architecture resource usage (e.g., size and time usage), and also documents it. The block library represents a reusable design library that may be applied to one or more projects.

#### *Model for a Functional Simulation*

The diagram must be more than just a picture of the application; it also must serve as the basis for a functional simulation to allow the designer to test out candidate algorithms, to vary model parameters in order to see the corresponding effect on performance, and to functionally prove out a design. This model becomes an executable specification of the application and represents the functional requirements of the application. The simulation environment should also be able to feed data and control signals to an application model, to allow the user visibility within the model to debug it, and to provide a way to capture the model output data and control signals for comparison and validation against application requirements. A simulation environment also allows development of test vectors that may be used to later validate an implementation.

At this level of development, the user constructs a block-based model of the application and, through simulation, proves whether the design meets functional requirements. The designer is operating here at a processing granularity of a block, which may perform a simple operation such as summation of inputs, or a more complex function such as a Fast Fourier Transform (FFT). In either case, the designer relies upon the assumption that the system has tested, validated, and documented each block before its use. The system thus expresses the application at an abstraction level in terms of the operations of the blocks, and not upon the inner workings of code that may implement the block. Reuse of components aids in design productivity.

### Architectural Model

The Architecture Model functions in a role similar to an Application Model in that it is a high-level summary of the target platform on which the application will execute, and depicts the configuration of the architectural parts. Ideally the Architecture Model is expressed in terms of a hierarchical block diagram with connections indicating the flow of data and control. The blocks in such a model may come from a library of architecture elements whose ports and behavior represent standard protocols, such as PCI, or the user is free to construct a new architectural element. Blocks may also have parameters, such as a clock rate for a processor block or bus speed for an interconnect element. Other application design attributes may be extracted from such a model, such as electrical power dissipation and cost estimates.

### Performance Analysis and Simulation

Analysis takes the Functional Model and maps it onto the Architectural Model. That is, it assigns the processing blocks of the first to the second. This assignment may be either a manual or an automatic process. In either case, for a mapping assignment, information from the two models is fused together, along with the effects of the overhead of an operating system in which the software executes, and simulated to estimate the performance of the execution of the application on the architecture. Performance here means summarizing such attributes as execution time, memory or resource loading, and power utilization.

The Analysis is based upon the information obtained from the Functional and Architectural block libraries that characterize how each of the Functional blocks executes on the Architectural elements. The effects of an operating system in terms of the execution overhead and memory footprint are incorporated into the analysis model. Analysis techniques, such as Rate Monotonic Analysis, custom device analysis, or simulation-based dynamic analysis are then applied to the application and compared against the characteristics delineated in the Requirements. With multiple mappings or architectural candidates considered, a matrix summarizing the system performance attributes can be constructed. This matrix allows the designer to weigh the factors and performance tradeoffs between the attributes of each candidate and make an informed decision in selecting one over another.

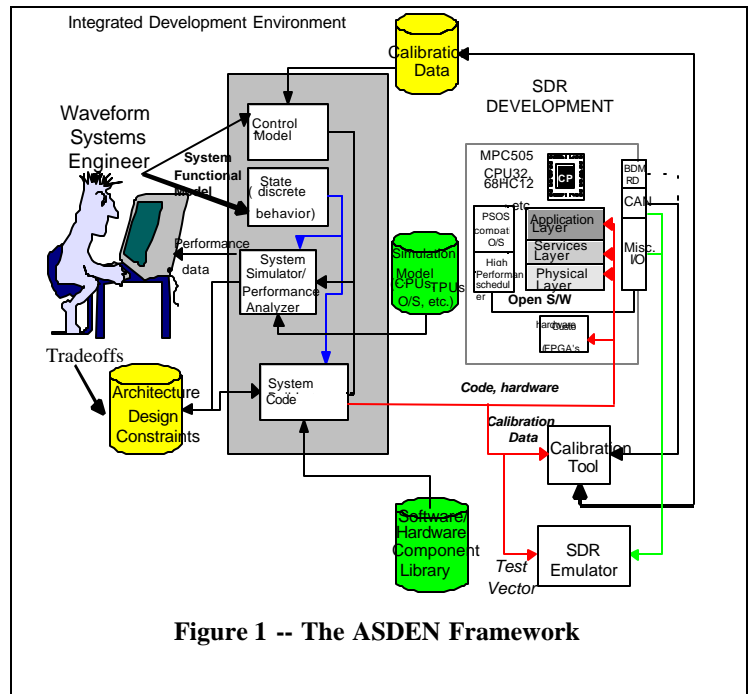


Figure 1 -- The ASDEN Framework

### Compile-Link-Load Phase

With the selection of one or possibly several candidates, the application design is passed on to implementation phase processing. This phase involves architecture-dependent tools that support the elements specified in the Architecture Model, the source code representing the block processing from the Functional Model, and the software architecture of the operating system controlling the application. The ASDEN Code Generation System (CGS) can support a number of standard source languages, such as C, Ada, or processor-specific assembly languages. In any case, the code generated for a specific architecture is tuned for that target. At this point, processor-specific tools, such as cross compilers and target debuggers, are used to create object modules from source code and to download them to target development boards. For those functional components that have been mapped to off-the-shelf or custom HW, FPGA and ASIC design tools are used to specify the architectural element.

### Target HW Based Debugging

After functionally proving the application in a simulation environment, applying performance analysis to the target, and generating target platform-specific load modules, the designer is then ready to validate the application running on the architecture instance. At this point, the designer needs a debugger tool to impart visibility and control into the operation of the application. The following subsections describe briefly some of the key features of an Integrated Development Environment (IDE) with high-level models as a starting point for capturing the application. Simply expressed, these features allow the user to validate the application behavior on the target architecture instance at the same level of abstraction, visibility, and control offered in the functional simulation environment. This is the primary mode of debugging, with the ability to look inside the

model blocks as a feature held in reserve. For example, if a Finite Impulse Response Filter (FIR) is specified as a block, the desired debugging level of the application assumes that the filter is implemented correctly and deals with inputs and outputs of the block, not with the inner workings of the code implementing the block.

### *Symbolic References to High-Level Model*

When executing the application on the target architecture instance, the designer wants to first verify the operation at the same level of abstraction used for modeling and simulation: at the block level. This capability depends upon visibility and control at the block level—i.e., to be able to examine the data and control streams between functional blocks as implemented on the target architecture instance. The designer also wants to set breakpoints and to single-step the chunks of code corresponding to individual functional blocks. Source code, such as C language or statements or labels, is not appropriate at this abstraction level. Names of blocks and connections are a minimal supporting paradigm for breakpoints and data watchpoints. A point and click paradigm on the graphical model representation is a more desirable approach.

Whereas the primary need for debugging should be at the block level, a robust debugging system should also provide for visibility into the source code implementation and allow control to be exerted at that level as well. In other words, desirable is the capability to look inside the functional blocks. This would be used to trace borderline cases of behavior. Note this level of debugging would be one of last resort in relation to robust reusable algorithmic blocks, communication functions to tie the multi-processor network together, and modular operating system components to run the application. Because it requires operating at a different abstraction level, it may also involve designers other than the originator of the system-level diagram (for example, hardware/software embedded designers familiar with bit level implementation issues).

### *Static and Dynamic Debugging*

“Static” refers to examining the state of the application after it has executed for a period of time or to a specified point in the processing and then halting the operation of one or more elements. This approach allows the designer to examine the state of the data and processing elements to verify their operation.

“Dynamic” refers to examining data and control streams as the application is executing. Ideally this would be done without affecting the performance of the application; but often it imposes some form of overhead or change to the application, such as running with instrumented versions of communication or operating system components.

### *Data Visualization*

Directly related to data visibility is the issue of data visualization, i.e., presenting the information meaningfully. This approach assumes the form of simple tabular presentation for generally small amounts of data, two-dimensional plots for some time-dependent data, multi-dimensional plots for multiple-

channel data, and application-specific plots, such as eye diagrams for communication systems.

## **Waveform Development Tools within ASDEN Flow**

The basic processing flow for waveform applications is well suited to the ASDEN structure, which provides for initial separation of functional design from target platform characteristics. As described in previous sections, via the use of libraries mapped onto many potential target HW architectures, the ASDEN fuses function and architecture information to allow for rapid HW/SW Co-Design tradeoffs based upon the performance metrics obtained from the mathematical rigor of Rate Monotonic Analysis techniques and confirmed with performance simulation. When the final target configuration is selected, source code representing the implementation of the application processing and surrounding SW architecture is then automatically generated for all the selected target's processors, programmable peripherals, and other architecture elements under SW control.

The following sections outline how FORESIGHT SYSTEMS INC. has instantiated the ASDEN methodology for prototype IDE targeted to Waveform design. While specific tools are called out for each phase, the ASDEN interfaces are open. Adaptation to another best-in-class tool for a specific design phase is a matter of interface creation. In summary, the main current tools used are:

- **Foresight Co-Design** from Foresight Systems - This is a high-level system design tool for block diagram modeling and simulation. It supports requirements tracking and simulates system performance and resource usage.
- **MATLAB, Simulink, Stateflow** from the MathWorks, Inc. - This is a numerical algorithm development and system-modeling tool with customizable code generation capability.
- **Rate Monotonic Analysis** -- A customized package of analysis tools based upon work done on the PERTS project at the University of Illinois, Urbana-Champaign.
- **Commercial Development Tools** - An expandable set of compiler, linker, and debugging tools specific to target HW. ASDEN defines a generic interface to allow such tools to plug-in. Libraries of parameterized performance information are also built by using these tools on supported target architectures.
- **Foresight Systems Developed Interface/Translation Tools** - As part of the implementation, Foresight Systems uses in-house parser and database tools to extract information from other tool output to provide the "glue" that makes the tools of each design phase work together as a complete IDE.

### Requirements Capture

Telelogic's DOORS tool is a market-leading requirements tracking and analysis tool. Both of the main modeling tools used in the ASDEN implementation, Foresight Co-Design and Simulink, support links to DOORS. The design flow uses links to functional and architectural requirements between the three tools. In this methodology implementation, with Foresight Co-Design playing the central performance simulation and

validation roll, it will also act as a central modeling coordination and linkage tool to bring together results from design phases.

## Functional Model

The previous sections emphasized that the basic purpose of the functional model within ASDEN is to capture the processing being performed by the application. This capture includes data as well as control processing of the applications. In summary, the functional model is created in a high-level modeling tool, which provides simulation capability so that the application can be tested and verified to meet system level requirements. By working at a high level the designer is able to concentrate on the nature and method of the task to be achieved. The following sub-sections describe Simulink's capabilities in this area.

### *Using Simulink for Waveform Processing*

Simulink provides a robust graphical block diagram environment to design and test the data processing of systems. Blocks represent the processing to be done, whereas connecting lines represent the flow of data between the functional blocks. Armed with both the basic Simulink block library and reusable application-domain-specific block libraries, creation of an application seems as simple as a block drag and drop from a library, and a connect-block signal ports drawing paradigm.

With the integration of Simulink and the MATLAB command environment, parameterization of an application at the system level is easy via the parameterized block library. Sets of parameter values can be defined within the MATLAB workspace and symbolically referenced in the Simulink block parameter values. The values can then be changed in one place, the MATLAB workspace, with automatic propagation of the effect throughout the Simulink model. This processing allows the designer to explore the design space easily and efficiently.

Because the model is rendered in a diagrammatic form with labels and user annotations (comments within the diagram), this is a self-documenting and executable specification of the functional system requirements.

### *Using StateFlow for Waveform Control*

StateFlow is a graphical State Transition Diagram (STD) tool used for the creation of finite state machines. It is integrated with Simulink, which means that StateFlow blocks can be incorporated within Simulink diagrams to exchange data and otherwise interact with the Simulink model. The tool allows for hierarchical design of state transition diagrams and can associate actions with the transitions and states. The actions can be defined as block outputs and function calls activated upon entry, exit, and remaining within a state or transition.

With these key features, the control processing of a waveform application may be represented as STDs to interact and exert control over the waveform signal processing. These are suitable for application operating mode control and man-machine interfaces for parametric control.

### *Create Reusable Waveform Library*

A key to the use of block diagram-based tools is a rich and user-extensible library of building blocks from which to create

models. Simulink comes with a basic block set of about 80 blocks providing manipulations that range from adding signals together to solving partial differential equations used to represent the State Matrix approach of control systems. Supplemental block sets specific to some applications domains may be purchased. Such block sets provide extensive libraries of blocks implementing transforms and data representations found in textbooks. Simulink is an open system in that it provides a documented way for the user to add custom blocks written either in the MATLAB programming language (M-Files) or in more traditional C source code, called "System Functions" or "S-Functions."

### *Simulation and Verification within Simulink*

Simulink provides a simulation engine to execute the blocks in a model according to data dependency requirements expressed in the model. Simulink maintains sample rate information specified within the model and can handle multi-rate systems. It provides for the building of models that respond to trigger and enable signals, and thus allows the integration of several execution activation paradigms and their functional verification via simulation. This environment provides the basic testing and validation method for the functional correctness of waveform applications.

### *Functional Model Information Extraction*

ASDEN can support several modeling paradigms with different models of execution or semantics for the time and sequence in which the model blocks will execute. To enable these models to work together, they are put in a canonical form for further manipulation, analysis, and code generation.

For use with Simulink, much of the information needed for ASDEN purposes is available from the open architecture of the Real Time Workshop (RTW) tools that form the code generation system of Simulink. After RTW executes, the tool checks the structure of the model and analyzes the application to construct tasks to execute the model. The open design of the code generation system allows third parties to adapt the Target Language Compiler (TLC) processing of RTW to generate model information for ASDEN use. ASDEN uses the TLC to extract functional component information from the model. These components are then available for later processing in ASDEN during HW/SW co-design phases implemented in Foresight Co-Design based architecture models.

## Platform Architecture Model

The basic purpose of the Architecture Model is to represent the target HW platform characteristics in a high-level model that identifies those attributes of the model elements that affect overall system performance. The following section defines features of Foresight Co-Design that support such models. The key feature that makes Foresight Co-Design suitable for this task is the built-in resource model and usage paradigm. The tool is also open in the sense that it provides links to other OMI compliant simulators, external C code, and VHDL simulators.

### *Define Implementation Target Elements*

ASDEN builds the architecture model from reusable library elements in ways similar to that described previously for the functional model. The ASDEN implementation uses Foresight Co-Design to define new architectural components in reusable parts libraries that can be used in a drag and drop drawing paradigm to visually lay out and arrange the HW elements in a target platform. Examples of components include processors, memories, buses, protocol handlers, and peripheral devices. These elements manipulate the data, route the data through the architecture, and provide data streams and sinks.

### *Connectivity of Elements*

The Architecture Model defines what elements are available to implement an application, and to implement the connectivity of the elements. Using the same signal connection paradigm of the functional model, the designer uses connecting lines to represent the flow of data or control between the Architecture elements. The ASDEN mapping algorithms use these communication paths to route the application data between the functional model elements while Foresight Co-Design tracks the usage of the communication elements or resources.

### *Ties into Performance Characterization*

Performance attributes are associated with each of the library block elements and connecting lines. These parameterized features specify capacity, sizes, and rate information, which, in turn, specify how the model may perform.

### *Ties into Other Architectural Tradeoff Attributes*

ASDEN supports tracking and association of other features, in addition to performance attributes, with the architectural elements. For example, other factors such as cost, power, and size may be assigned to the elements and then aggregated for system-level totals on such factors. These totals are then available for inclusion in tradeoff studies of candidate architectures to implement the Application.

### Performance Analysis and Assignment

Performance analysis is the examination of the behavior of architectural elements interacting with executing functional elements and determining the loading and throughput of the architecture. Assignment is the process of obtaining elements of the functional model and mapping them onto the architectural elements. By the fusing of the models, use of analysis techniques with supporting Foresight Co-Design simulation, the ASDEN tools then determine the overall system rate and capacities of the assignment.

### *Rate Monotonic Analysis (RMA) and Simulation*

The customized tools of ASDEN performance analysis employ techniques of Rate Monotonic Analysis (RMA) in order to determine schedulability for core and/or peripheral devices. The basic paradigm is that of a set of tasks, with processing strings which run at a bounded rate or frequency, that compete for the use of limited resources. Traditionally these resources are processors; but the same techniques can be applied to other resources such as busses, memories, and I/O devices. The

approach assumes that a through analysis must consider the operational characteristics of each device. Past experience has shown that the unique behavior of some devices may justify custom analysis in order to identify the range of possible values.

Foresight Co-Design's resource modeling and usage simulation features are used to enhance and supplement the RMA findings via performance simulation. Such detailed simulation allows for data-dependent processing and behavior to be taken into account that can be less pessimistic than the enhanced RMA analysis. It also provides a way to investigate the functional and architectural design interactions to identify problem areas such as bottlenecks and to test design refinements.

### *Functional Element Assignment to Architecture*

As part of RMA, the ASDEN tools provide capability for automated assignment as well as for manual control of the mapping. The current automated approaches are based upon bin-packing heuristics. The user selects from bin-packing algorithms such as First Fit, Best Fit, and Worst Fit, where the "Fit" refers to the schedulability loading limits based on the task assignment to the processor.

A separate class of automated assignment algorithms performs what one might call "an exhaustive pruned search." That is, ASDEN systematically assigns the set of tasks to the set of processors. However, as the assignments occur, and upon analyzing schedulability, ASDEN terminates or prunes those branches of the search tree that violate schedulability. Research in this area indicates that the pruning action limits the number of examined combinations, and thus limits the combinatorial growth of the search time.

### Code Synthesis

Upon completion of the candidate architectures and functional element assignments, code is then synthesized for the target platform. This code must implement not only the processing of the functional model, but also the operating environment in which the application executes. ASDEN uses two approaches to complete the code for the application system. Each is described in the following two subsections.

### *RTW Provides Functional Code*

Real Time Workshop (RTW) is the built-in code generation system of the MATLAB/Simulink/StateFlow tool environment. RTW enables the user to automatically generate C code that is customizable for the target platform via the TLC tool. With parameterized TLC support created for each target platform and each compiler, one model may be used to optimize code for each target configuration.

### *MCT Integrates Functional Processing with SW Architecture*

The Model Conversion Tool (MCT) is a lexical analyzer, parser, and template-programming tool developed by Foresight Systems. ASDEN deploys MCT in several places in the tool framework, such as to read in model files. It also functions within the current prototype implementation of ASDEN to process the key attributes of the model structure extracted by TLC programming. The capability of MCT to model information and to program templates supports the code generation process

by embedding the block code of the functional model into SW architectures customized to target HW architectures. Application structure information from the model file is extracted to apply parametric changes to the template SW architecture. The goal is to free the high-level modeling tools and their code generators to focus on the block code, while the MCT supports the creation of the overall target-specific SW framework into which the block code is to be inserted. This SW architecture handles target-specific issues such as operating system (OS) interfacing, interrupt handling, and peripheral control. It also allows ASDEN to integrate models from different tools into a common framework.

### **Conclusion**

In this paper, Foresight Systems has described how the pre-existing and growing ASDEN technology would support development of a waveform development environment. ASDEN provides a comprehensive framework that delivers rapid refinement and faster implementation of selected solutions for waveform development technology via its functions in requirements capture, performance analysis, tradeoff analysis, code generation, debugging and optimization, and loading. This framework empowers the designer to meet challenges proactively and to coordinate hardware and software design with higher efficiency, accuracy, and faster delivery of tools or end products to target markets. On an individual level, ASDEN will deliver and accommodate the unique integration of toolsets for the specialized needs of each designer, whether in realizing the waveform development concept and supporting SDR architectures or in exploring additional value to be derived from each of the many disciplines of engineering, science, and project management.

### **REFERENCES**

- [1] *Modular Software-programmable Radio Consortium*, December 15, 2000, Software Communications Architecture Specification
- [2] JRS Research Laboratories Inc. *Design Methodology for the Advanced Systems Design Environment (ASDEN)*, 8 August 1997.
- [3] Orfali, Robert, Harkey, Dan, and Edwards, Jeri. 1997. *Instant CORBA*. New York: John Wiley and Sons.
- [4] Cook, Peter and Williams, Larry, *Software Architecture in Software Defined Radios: Past, Present, and Future*
- [5] Vortal, Mike, August 1997, *High-Speed Communications: A Modeling Approach*
- [6] The MathWorks, *Using Simulink, Target Language Compiler Reference Guide*
- [7] Foresight Systems, Inc. *Foresight Users' Guide*