Combining Foresight and MATLAB for Complex System Design

By Paul Stoaks
Foresight Systems, Inc.
pstoaks@foresight-systems.com

© 2002, 2001 Foresight Systems, Inc. All rights reserved.
Printed in the United States of America

Trademarks

The following are trademarks or registered trademarks of their respective companies or organizations:

Foresight, FS/Bridgeway, and FS/ Vis are trademarks of Foresight Systems, Inc. See our web site at http://www.Foresight-Systems.com.

MATLAB and Simulink are registered trademarks of The MathWorks, Inc.

UNIX is a registered trademark of The Open Group.

Microsoft, Windows, ActiveX, and Visual C++ are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Other product names mentioned in this document are trademarks or registered trademarks of their respective companies or organizations.

Introduction

Modern systems are very difficult to construct and verify due to their growing complexity. Multiple disciplines must come together in order to achieve the design goals. For modern, complex designs, model-based design provides a significant advantage, but a model-based design environment must support multiple levels of abstraction and mixed modeling domains. This document presents a methodology for combining Foresight and MATLAB to create a powerful, flexible environment for exploring the system design space and formalizing specification prior to implementation. In particular, it provides the ability to determine both the behavior and the performance of a system prior to implementation while providing for verification at all stages of the design process.

System Design with Foresight

Foresight™ provides a versatile and powerful system design and modeling environment. Its power and flexibility are due to its rich modeling language, powerful resource modeling capabilities, mature discrete event simulation engine, and flexible model integration APIs. The Foresight modeling environment is depicted in Figure 1.

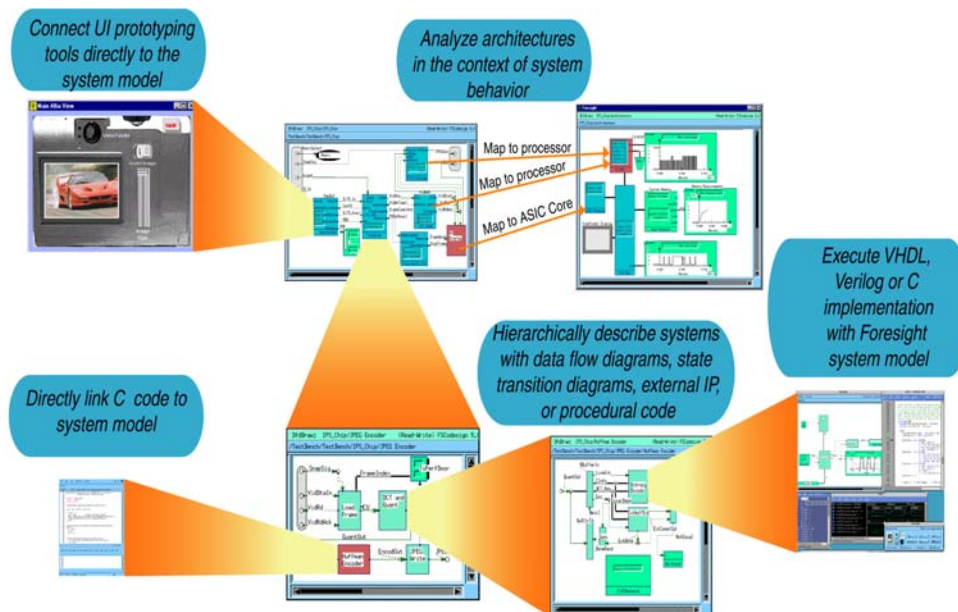


Figure 1: System Design Using Foresight

Modeling Language

Foresight's modeling language includes hierarchical state transition diagrams, data flow diagrams, and a procedural language, as illustrated in Figure 2, below. These may be hierarchically combined to form complete models of system behavior. The Foresight

modeling language aligns closely with the way engineers conceptualize their systems allowing them to quickly capture their system specification.

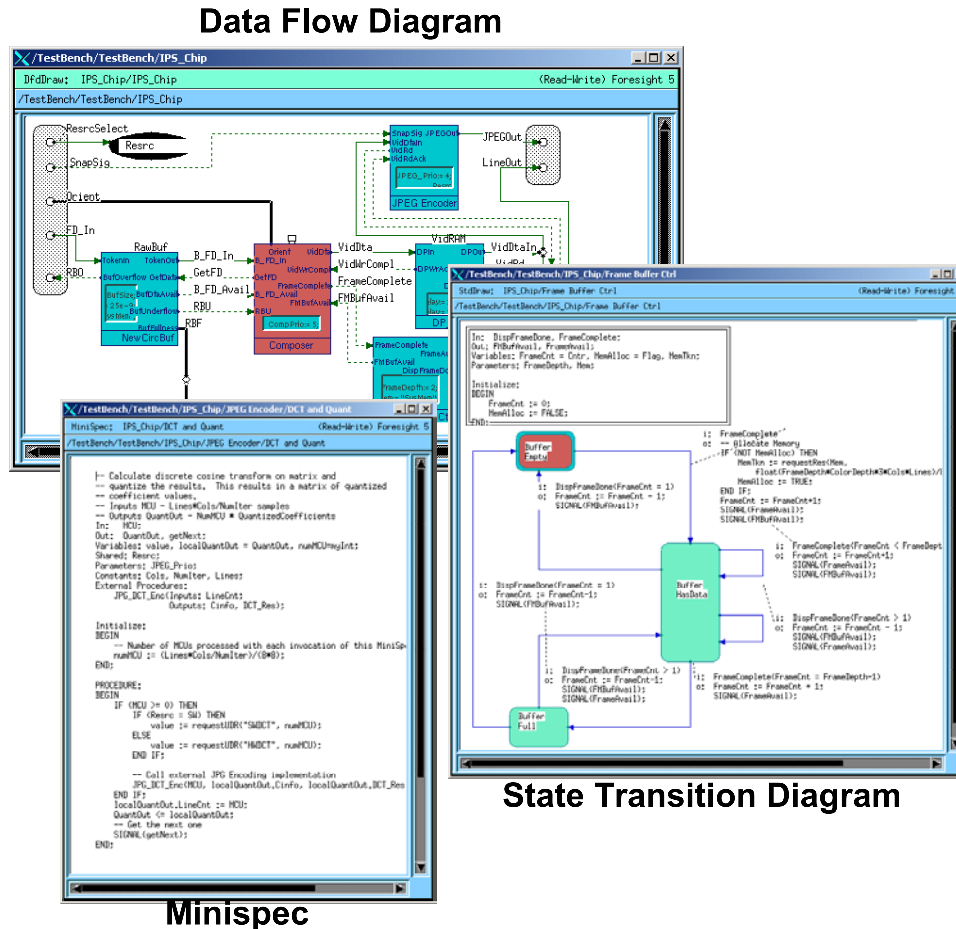


Figure 2: Foresight Modeling Language

Resource Modeling

Foresight's resource modeling capability is the key to enabling accurate performance estimates and system optimization. Foresight's library provides two different kinds of pre-defined resources, the Basic Resource and the Process Resource. The Basic Resource represents components that have some capacity that can be consumed, such as memory, a battery, etc. The Process Resource represents a component that can do work, such as a processor, bus, OS service, person, etc. The Process Resource has a rich set of parameters that control scheduling behavior, preemption, etc.

In addition, Foresight provides users with the tools to create custom resources with arbitrarily complex behavior. This extremely powerful feature facilitates the creation of very sophisticated resources that can accurately model scheduling, arbitration, and various kinds of analysis and data collection

When a functional block is mapped to a basic resource (either the built-in Basic Resource or a user-defined resource with basic resource behavior), it incurs a delay until the resource becomes available. When a functional block is mapped to a process resource (either the built-in Process Resource or a user-defined resource with process resource behavior) it incurs a delay until the specified amount of work has been completed.

It is through this mechanism of mapping functionality to resources that performance models are created in Foresight. Modeling the dependence of a functional model on system resources introduces delays into the system in a natural, realistic way that is superior to any static analysis methods or input-to-output delay annotation mechanisms. Resource contention and its impact on performance is modeled directly, as opposed to being abstracted to point-to-point delay functions.

In addition, alternative mappings of functionality to architectural components can quickly be explored in order to optimize the system. The hardware/software tradeoff is an example of alternative mappings of functionality to architecture that is facilitated by this methodology.

Discrete Event Simulator

The Foresight model is executable via Foresight's integrated Discrete Event Simulator (DES). The Foresight discrete event simulator is a true DES that incorporates a rich set of flow and process input sensitivity semantics. The richness of these semantics makes it possible to naturally model an extremely wide variety of systems.

The discrete event simulator can also be used to verify that system requirements are being met. Library elements allow for flagging violated requirements (such as missed deadlines, assertions, etc.) at simulation time so that they become immediately obvious. In addition, many library elements allow for data input and visualization output. Monitors can be placed on flows and processes and the data recorded to text files for later analysis. Powerful debugging capabilities are integrated into the simulator to assist in arriving at a correct model.

Model Integration APIs

Foresight provides a number of flexible mechanisms for connecting Foresight simulations to other environments. These APIs make Foresight an open model integration framework.

The External Call Interface enables the simple creation of user defined functions in 'C' or 'C++'. These may be called directly using the Minispec language from within State Transition Diagrams or Minispecs. User defined functions are synchronous in that the simulation does not progress until the function returns. They can be used to perform virtually any data manipulation function desired.

The Bridgeway interface enables the straightforward creation of asynchronous IPC links between Foresight models and other applications. Both peer-to-peer and client-server configurations are supported. Co-simulation interfaces have been created to HDL simulators, instruction set simulators, and a number of other modeling and simulation environments. Bridgeway has also been used to connect hardware and software to perform implementation-in-the-loop simulation and verification.

Finally, the Foresight ASCII database format allows for straightforward access to, or translation of, Foresight model databases. Conversions to XML, HTML, and SVG have been created directly from the ASCII database format.

Design Flow with Foresight and MATLAB

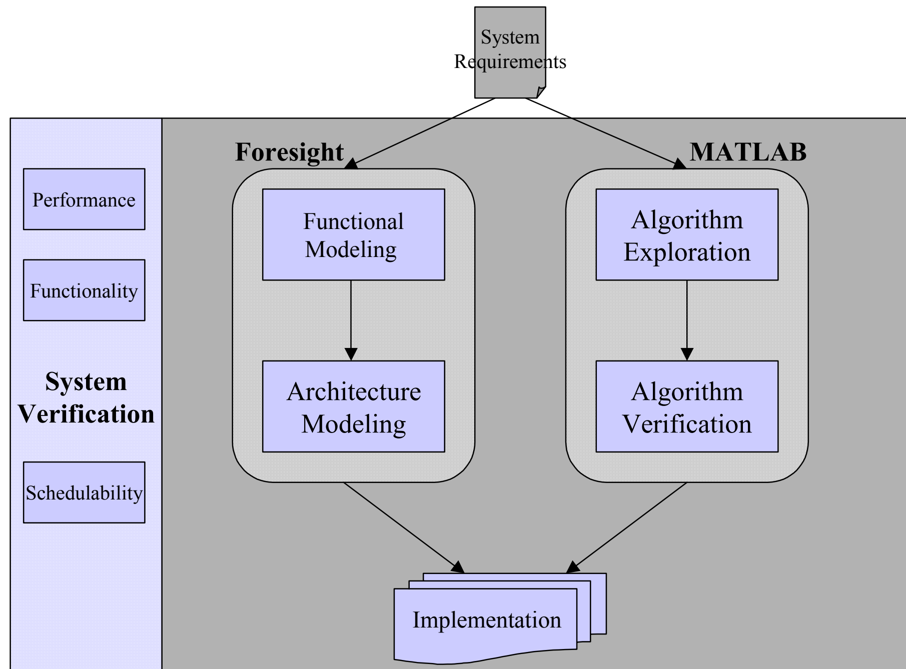


Figure 3: Design Flow

A rough depiction of the system design flow is shown in Figure 3. In this design flow, design exploration starts with system requirements identified during requirements analysis[1]. Foresight is used for overall system design, while MATLAB is used for algorithm design. MATLAB and Foresight are integrated for simulation in order to enable system verification.

Each design component is explored below.

Functional Model

First, a Foresight model is constructed. The model should specify the system functionality and control behavior and address the following issues:

- Hierarchical partitioning
- Data and control flow
- Data types (Data Dictionary)
- Component interfaces

[1] FS/RQIF-DOORS is an add-on to Foresight that provides an interface to the DOORS requirements tracking system by Telelogic. This interface enables requirements trace-ability between the requirements database and the system model, as well as mechanisms that allow automatic verification that the system meets requirements.

During this phase MATLAB is used to begin exploration of the data processing algorithms required by the system. Ideally, the algorithm partitioning in MATLAB should follow the functional system partitioning in Foresight such that there is a one-to-one mapping between Foresight components and MATLAB algorithmic components at some level of abstraction.[2] This facilitates the later integration of the Foresight model with the MATLAB model.

Algorithm Model

As algorithmic components are defined in MATLAB, they may be verified independent of the system model in the MATLAB environment. They may also be connected to their counterparts in the Foresight environment allowing them to be exercised within the context of the Foresight model. This allows the realization of the ideal shown in the diagram, which is that system verification be possible continuously throughout the design process. The function and control model can be verified (at a high level) before the creation of algorithmic components, algorithmic components can be unit tested, and the system model with algorithmic components integrated can be verified to ensure overall system integrity.

Architecture Model

Next, an architecture model is constructed (or reused, if available) to represent the resources in the system. Ideally, the architecture model should be layered to match the actual system architecture. In Foresight, the architecture model is a collection of interdependent resources that make up the platform upon which the functionality will be deployed, and may consist of software services, busses, processors, memory, etc.

Performance Model

Functional model components are then mapped to the architectural components that implement them to form a performance model. A given architecture may support multiple implementations for any given functional component. For instance, an algorithm may run on processor A or processor B or be implemented in an ASIC. As a result, it is often desirable to evaluate multiple mappings of functionality to architecture. In fact, there may be multiple architectures that might be considered for a given design.

Analysis and Verification

Architectural components can be monitored for load, instantaneous capacity, etc. and monitors can be placed on processes, flows and resources in order to acquire data for post-simulation analysis. Execution of the model yields the behavior of the system with consideration of the resource constraints present in the system. Algorithmic behavior can now be evaluated in the context of the actual timing present in the system. This often exposes order dependencies due to out-of-order execution as a result of scheduling realities and resource availability.

[2] A single Foresight component may map to a more complex MATLAB call tree. It is not necessary for there to be a Foresight component for every MATLAB procedure or function call within the component. This supports the reality that MATLAB will model the behavior of many blocks at a finer level of granularity than is required in Foresight.

The system can be tuned in order to meet the functional and performance requirements. The "knobs" available for tuning include component parameters and functionality-to-architecture mappings (including hardware/software partitioning.) In a short period of time, many different configurations can be explored.

As implementation for system components becomes available (in HDL, software, hardware, etc.), it can be plugged into the system model for implementation-in-the-loop testing. This can occur much earlier than would ordinarily be possible with traditional integration and co-verification mechanisms. This is made possible through the Foresight co-simulation interfaces to HDL and ISS simulators, as well as the integration APIs already discussed.

Note that verification and tuning need not wait until everything is complete. Model-based system design can occur in an iterative manner where gross performance estimates are obtained very early to determine the portions of the system that are critical for success. Rapidly building a high-level model of the functionality and mapping it to a high-level model of the architecture accomplishes this. This model can then be analyzed and iteratively refined to the level of detail described above.

Integrating the Flow

There are a variety of methods available to the user for connecting MATLAB algorithmic components into a Foresight model. If the approach described above has been taken, then each Foresight component with algorithmic content has a corresponding MATLAB component. This allows the Foresight External Call Interface to be used to call the MATLAB-described functionality from the appropriate Foresight component.

Foresight External Call Interface

The Foresight Mini-spec language, can access functions written in 'C' or 'C++' via the External Call Interface. For example, consider a routine that converts a string to an integer using the 'C' atoi() library call:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// Include C++ version of fstokens.h
#include "fstokens2.h"

extern "C" {
  // c_atoi() returns the integer specified by the given
  // string using the 'C' atoi() function.
  DllExport FS_INTEGER * c_atoi(FS_STRING *s);
}

FS_INTEGER * c_atoi(FS_STRING *s) {
  static long rtn;
  rtn = atoi(s->str());
  return &rtn;
} // c_atoi()
```

The include file "fstokens.h" is found in the foresight/include directory in the Foresight installation location and declares the datatypes used in the External Call Interface. The "fstokens2.h" include file is a more recent, C++ addition that provides typesafe wrappers (using templates) for the Foresight data types. It is this latter version that will be used in these examples.

The source files for the user's routines are compiled and linked into a DLL or shared library, the location of which is provided to Foresight via a foresight.ini file in the model directory. The functions declared therein can then be called from a Minispec as shown below:

```
-- ATOI -
-- Accepts an input flow of type 'string' and produces an
-- output flow of type 'integer' which is that string converted
-- to an integer.  Relies on external call c_atoi() which uses
-- the 'C' library atoi() function.  Note that no error checking
-- is performed.
In:  s_in;
Out: i_out;
External Functions:
      c_atoi(Inputs: string) RETURN integer;

PROCEDURE
BEGIN
      i_out <= c_atoi(s_in);
END;
```

That's all there is to it. The External Call Interface is very simple to use and can call virtually any functionality that has a C API. All Foresight data types except the *alternative* are supported. It is perfect for accessing the MATLAB engine functions required to interface to MATLAB described components.

A Simple MATLAB Example

On UNIX, communication with the MATLAB engine occurs via the routines in the MATLAB engine library. On Windows, the MATLAB engine is accessed via ActiveX. In our examples and discussion, we'll be using Microsoft Windows and accessing the engine through ActiveX. Since this is the more complex of the two mechanisms, UNIX users should have no trouble mapping this to their platform.

As a simple example, we are going to construct a band-pass filter reusable component by interfacing to MATLAB[3]. The testbench for our component is shown below in Figure 4.

[3] The example model is packaged with this document and is available from Foresight Customer Support (support@foresight-systems.com.)

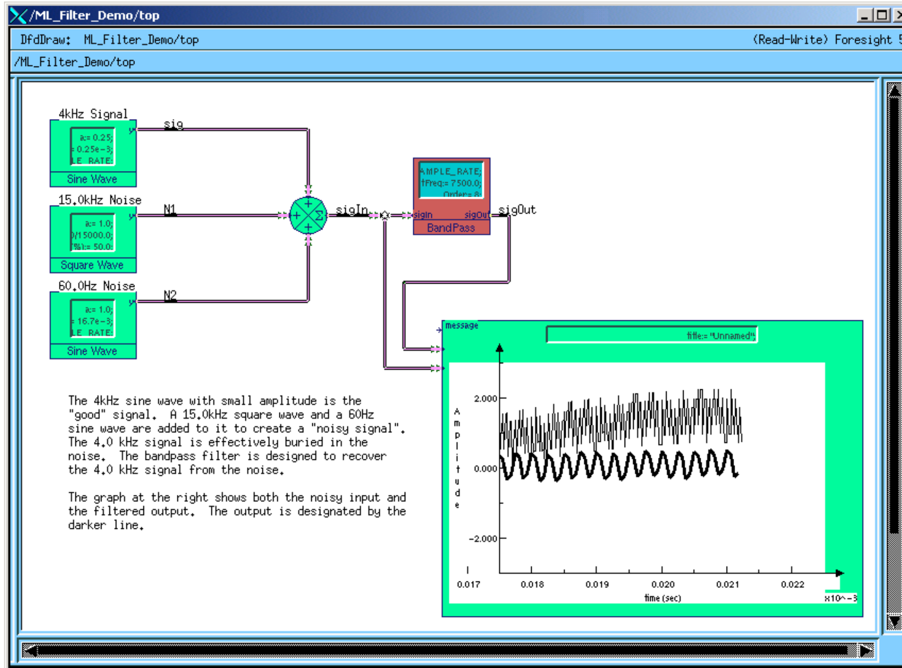


Figure 4: Band-pass Filter Testbench

The definition of the *BandPass* reusable is a data flow diagram and is shown in Figure 5, below.

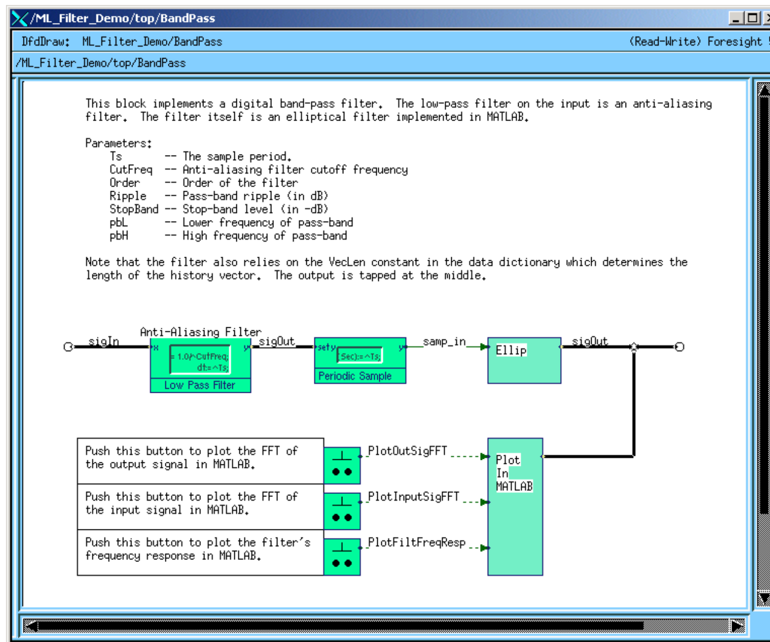


Figure 5: Band-pass Filter Component Definition

The heart of the filter component is the *Ellip* specitem, which is a minispec that calls MATLAB functions for the algorithmic work. Note that in addition to the filter functionality, some buttons have been added to graph various interesting data using MATLAB.

External Function Definitions[4]

The external function definitions are contained in a VisualC++ project with the Foresight "example_model" directory. There are three main "modules":

- dll_entry.cpp – Contains the definitions of the external functions that are the MATLAB interface. These include *MATLAB_Execute()*, *MATLAB_PutFullMatrix()*, *MATLAB_GetFullMatrix()*. These are examples and do not constitute a complete interface to the functionality available from the MATLAB ActiveX server. See MATLAB_App.h for the complete list of services. See the MATLAB External Interfaces manual for more information on how to use these functions.
- general.cpp – Contains the definitions of some frequently used external functions.
- MATLAB_App.(h,cpp) – Contains the definition of the MATLAB_App and MATLAB_Eval classes which specify the interface to the MATLAB ActiveX server. Using ActiveX can be tricky for the first-timer. Most of what is required for MATLAB use is implemented in the MATLAB_App class, so please look there for comments and examples.

The *ExternalFunctions* Visual Studio workspace compiles and links these into a DLL that can be called from Foresight. The definition of *MATLAB_Execute()* is shown below as an example. *MATLAB_PutFullMatrix()* and *MATLAB_GetFullMatrix()* are slightly more complicated due to the need to manipulate and translate matrices.

```
FS_STRING *MATLAB_Execute(FS_STRING *cmd_string) {
  static FS_STRING rtn;

  CString rtn_cstr =
      MATLAB_App::Instance()->Execute(cmd_string->str());

  rtn = rtn_cstr;

  return &rtn;
} // MATLAB_Execute()
```

For examples of calling these functions, please look at the *Ellip* minispec in the example model.

Output

In addition to the output shown in the Strip Chart in Figure 4, various MATLAB graphs can be obtained by pressing the buttons in the BandPass component. These are shown in Figure 6, below. The *Plot in MATLAB* state transition diagram in the *BandPass* Foresight component illustrates how the Foresight user can effectively employ MATLAB's powerful data analysis and graphing functionality via the ActiveX interface.

[4] Note that these are illustrative examples and may need further modification or completion for a specific purpose.

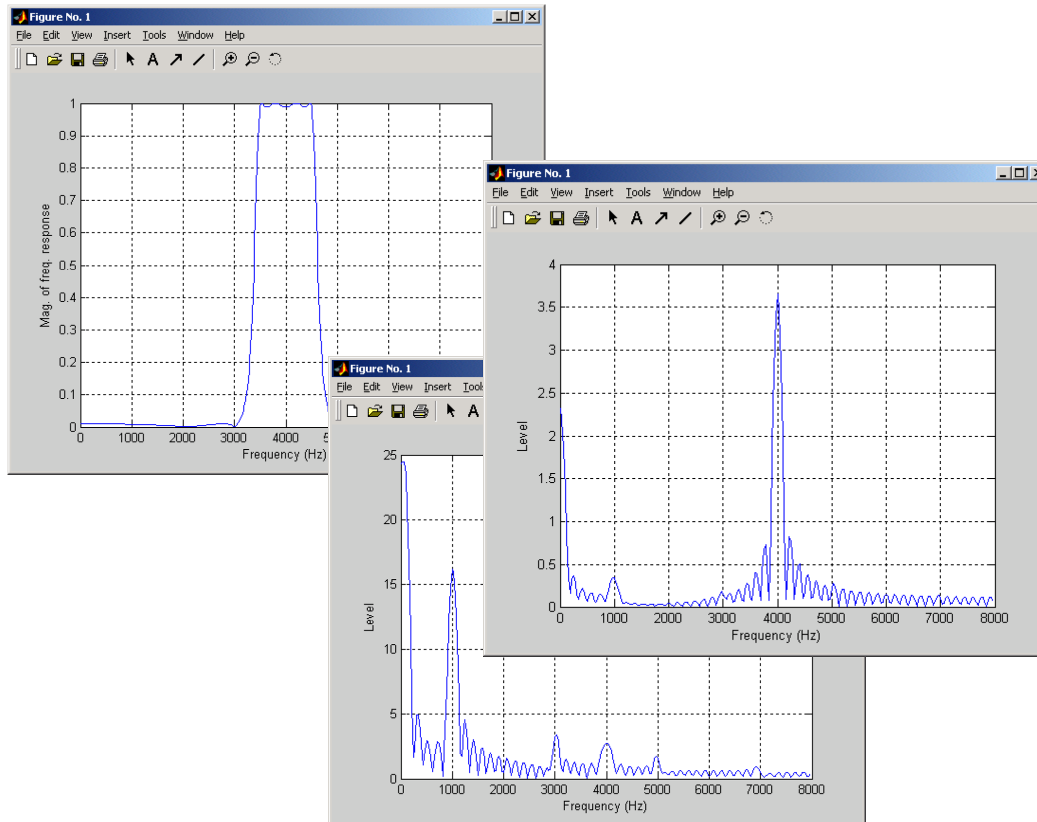


Figure 6: MATLAB Output

Getting Results

In order to understand the kinds of results that are possible with the combination of Foresight and MATLAB, consider the following example.

SCA Compliant Software-Defined Radio

The Software Communications Architecture (SCA) specification, developed by the Joint Tactical Radio System (JTRS) program office describes a radio device architecture that consists of a "waveform" and a "radio platform". The waveform is defined by software and is deployed on the radio platform which consists of hardware, OS, and other services. An objective of the effort is that waveforms are portable to multiple platforms, allowing a great deal of flexibility in radio configuration, upgrade-ability, and interoperability among radios used by the armed services.

Waveform Model

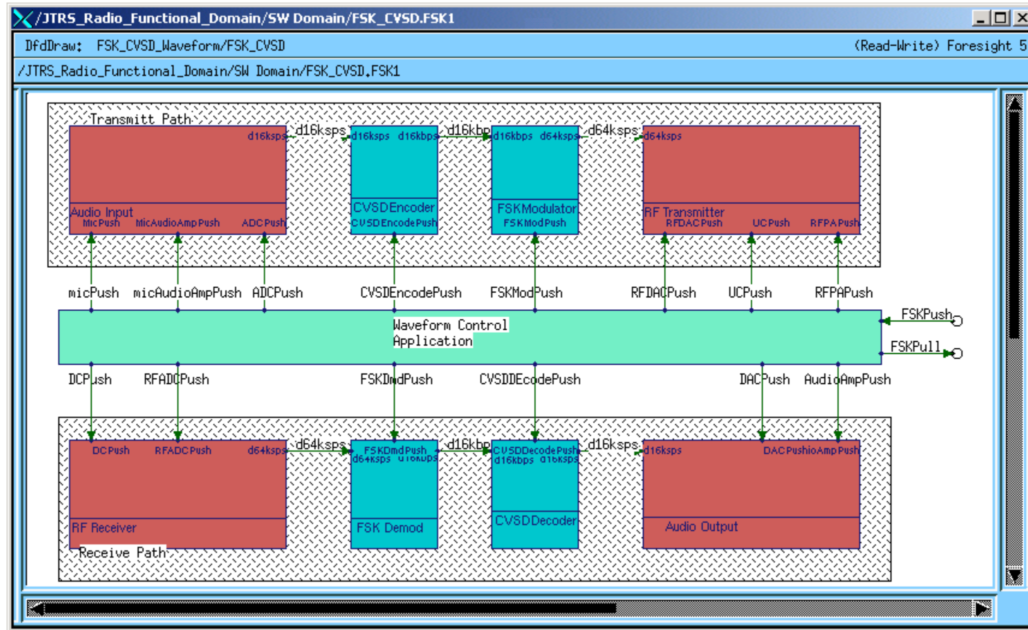


Figure 7: FSK Waveform for SCA

As per the design flow discussion above, the functional model of the system (the waveform) was modeled independently of the architecture (the platform.) The waveform model is shown in Figure 7. The transmit path is from left to right along the top, the receive path goes from left to right along the bottom. The Waveform Control Application represents the system infrastructure that controls the configuration and behavior of the radio, which is built on CORBA.

Figure 8 shows the definition of the CVSD Encoder component. Each component is initially constructed with sufficient fidelity such that the behavior and performance impact of the system and the CORBA infrastructure can be examined. The InPort and OutPort handle the interfacing between sequential components in the waveform. The PushPort handles the interfacing of the component to the system configuration services.

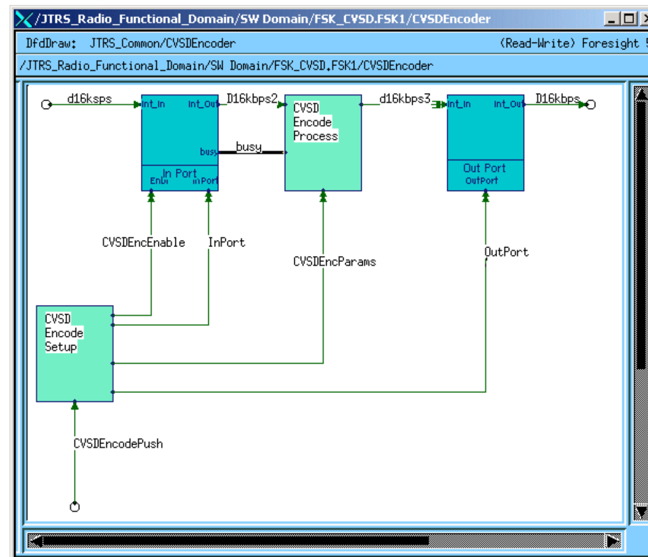


Figure 8: CVSD Encoder Block

Radio Platform

A simple example architecture, created for performance modeling, is shown in Figure 9. A more complete platform model would be layered and contain software services as well as hardware services. This simple example platform, however, provides excellent, first-order performance estimation results and suffices for our example. The platform consists of a bus, two general-purpose processors, an RF section, and an audio section. The RF and audio sections are further decomposed into their constituent components. The general-purpose processor not only models the microprocessor performance but also contains basic scheduling functionality such as would be found in an RTOS.

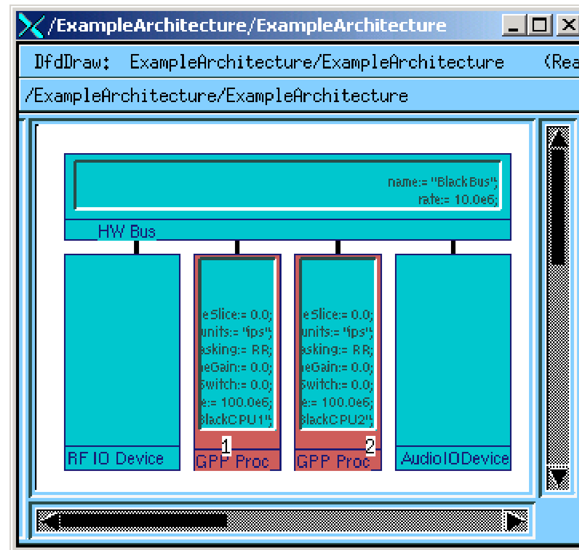


Figure 9: Simple example platform for SCA

The mapping of functional components from the waveform model to the resources defined in the architecture model is accomplished via parameters provided to the components through the system configuration infrastructure, just as they would be in the actual radio implementation. In other kinds of models, it is frequently the case that the mapping is provided in Foresight parameters on the components themselves. The parameter values may also come from files making it very simple to change configurations between simulation runs without modifying the Foresight model.

In this example, two instances of the waveform will be executing on the one platform.

When this model is executed, the following analyses can be performed.

Processor Utilization

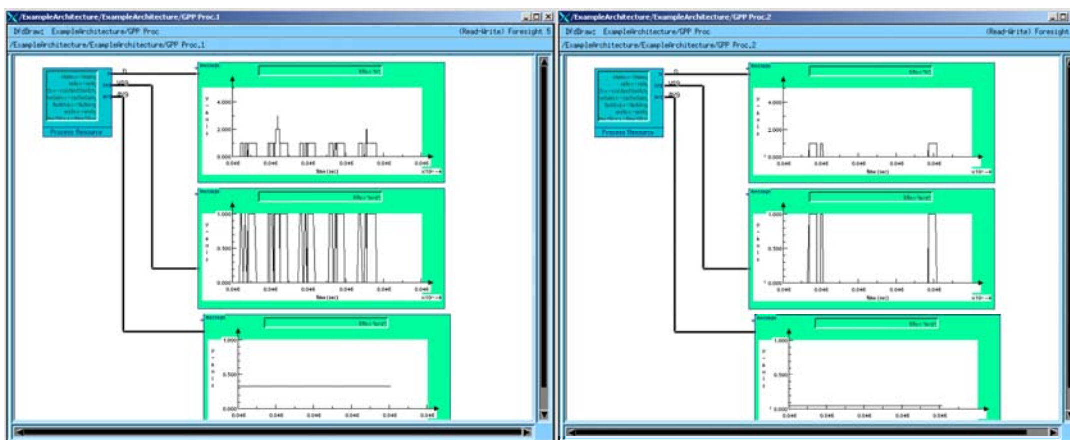


Figure 10: Processor Utilization

Figure 10 shows the processor utilization for the two general-purpose processors in the architecture with one waveform transmitting and the other receiving. Bus utilization can be monitored in the same way.

Queue Utilization

Queues throughout the system may be modeled and monitored to determine their utilization in the presence of system resource constraints and data rates. Figure 11 shows the monitoring of the input port queue during execution. The monitor data may be recorded to files for post-simulation analysis. Foresight allows any process or flow in the model to be monitored and recorded. Monitor data can be analyzed using the FS/Vis tool, or exported in comma-delimited format for import into Excel or other graphing and data visualization tools for further post-processing.

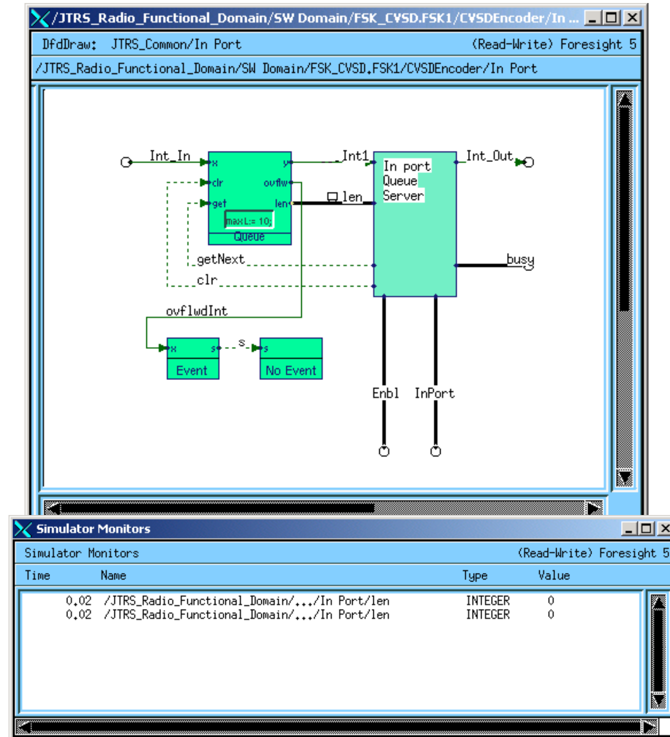


Figure 11: Monitoring Queue Length

Resource Contention

Resource contention may be analyzed using the Foresight Visualizer capability. Figure 12 shows the kind of timeline information that is available when using Foresight's Visualizer, FS/Vis, to analyze monitor data. Process activity and resource contention can be quickly identified using these facilities.

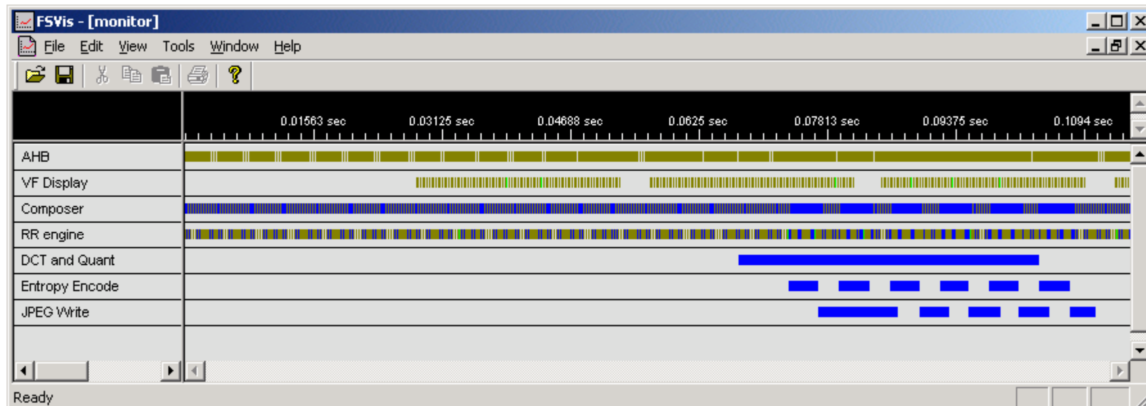
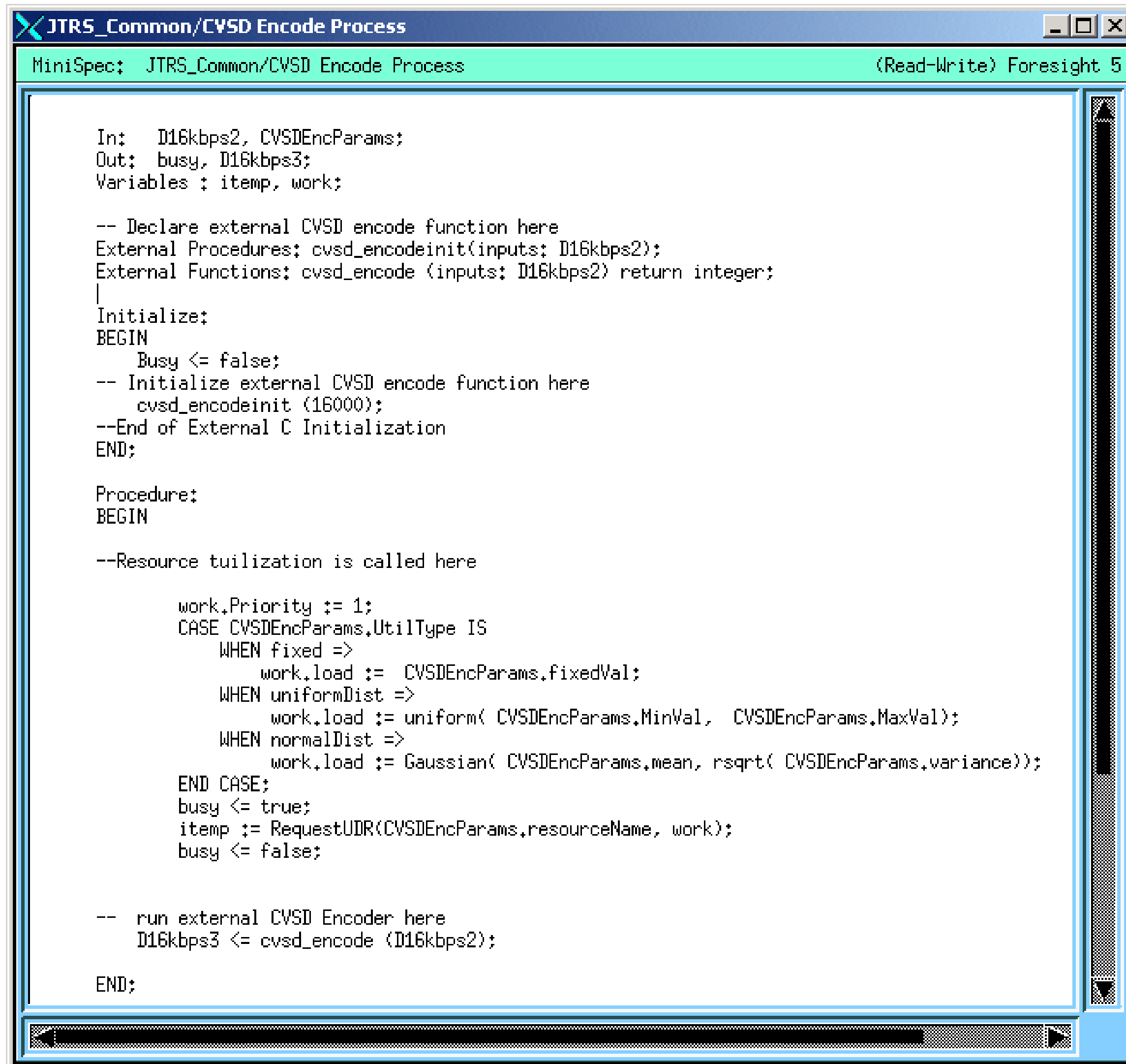


Figure 12: Foresight Visualizer

Algorithmic Behavior

Linking MATLAB models or other implementation into the model via the External Call Interface allows algorithm behavior to be monitored within the context of the larger system model. In this model, the CVSD Encoder algorithm is integrated with the Foresight model using the External Call Interface. The CVSD Encoder Minispec component is shown in Figure 13.




```
In:   D16kbps2, CVSDEncParams;
Out:  busy, D16kbps3;
Variables : itemp, work;

-- Declare external CVSD encode function here
External Procedures: cvsd_encodeinit(inputs: D16kbps2);
External Functions: cvsd_encode (inputs: D16kbps2) return integer;
|
Initialize:
BEGIN
    Busy <= false;
-- Initialize external CVSD encode function here
    cvsd_encodeinit (16000);
--End of External C Initialization
END;

Procedure:
BEGIN

--Resource tuilization is called here

        work.Priority := 1;
        CASE CVSDEncParams.UtilType IS
            WHEN fixed =>
                work.load :=  CVSDEncParams.fixedVal;
            WHEN uniformDist =>
                 work.load := uniform( CVSDEncParams.MinVal,  CVSDEncParams.MaxVal);
            WHEN normalDist =>
                 work.load := Gaussian( CVSDEncParams.mean, rsqrt( CVSDEncParams.variance));
        END CASE;
        busy <= true;
        itemp := RequestUDR(CVSDEncParams.resourceName, work);
        busy <= false;


--  run external CVSD Encoder here
    D16kbps3 <= cvsd_encode (D16kbps2);

END;
```

Figure 13: CVSD Encode Component

Note the declaration and calling of the CVSD Encoder algorithm and the resource utilization calls.

In addition, the FSK Modulator is fully defined in Foresight. As a result, it is possible to view the modulated signal produced by the waveform. Figure 14 shows the waveform output.

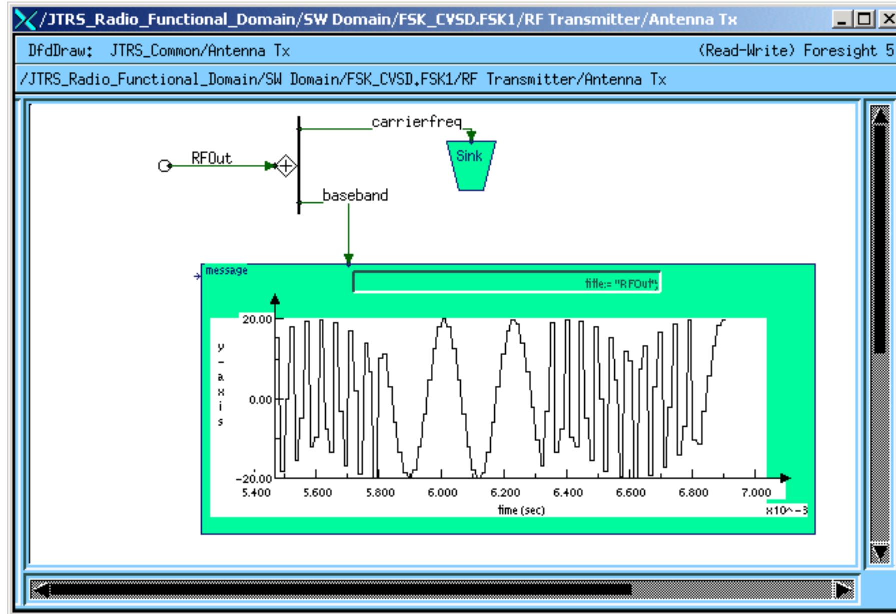


Figure 14: Waveform Output Signal

System Verification, Debugging and Optimization

Finally, as a result of being able to link a high-level system model, algorithmic models, and implementation together, it is possible to intelligently approach system verification, debugging, and optimization. Test stimulus and scenarios can be executed in the system model in order to verify performance and functionality. Unit tests for algorithms and implementation can be created from stimulus produced by the system model. Problems found can be more easily debugged because of the visibility of data and the debugging capabilities in Foresight. Sensitivity analysis and system optimization can be accomplished by manipulating parameters in the model.

Summary

In this paper, we have shown how Foresight and MATLAB can be used together to model the behavior and performance of complex systems and how this enables design verification at all stages of the design process. The following items were described and illustrated:

- A design methodology that provides for multi-level-abstraction, mixed-domain modeling, simulation, analysis, and verification.
- Details of how to accomplish interfacing Foresight and MATLAB to accomplish this methodology.
- The results that may be obtained from using this methodology.

This paper has focused on interfacing Foresight with the MathWorks MATLAB tool. Another white paper will cover the connection of Foresight with SimuLink in a co-simulation arrangement.

Bibliography

Foresight Mini-Spec Language Reference. Foresight 5 ed. McLean: Foresight Systems, Inc., 2000 - 2002. Originally published by Nu Thena Systems, Inc., 1998 -1999.

Foresight Users Guide. Foresight 5 ed. McLean: Foresight Systems, Inc., 2000 - 2002, Originally published by Nu Thena Systems, Inc., 1990 - 1999.

MATLAB, The Language of Technical Computing, Version 5 of A*pplication Program Interface Guide*. Natick, MA.: The Math Works, Inc., 1996 - 1998.