Frigg: A Simulation Environment for Multiple-Processor DSP System Development

Jeffrey C. Bier     Edward A. Lee

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720

ABSTRACT

Digital signal processing applications often require extremely high computation and I/O rates, combined with low system cost. To address these requirements, system designers often turn to custom multiprocessor architectures based on programmable digital signal processors (DSPs). Unfortunately, there are very few tools available to support the design and development of hardware and software for such custom architectures. While simulators exist for all programmable DSPs, there has been no readily available way for developers to simulate the interaction between a programmable DSP and other digital hardware, or between multiple DSPs (which may be different types). This lack of flexible simulation capabilities has often meant long delays in hardware development, and the postponement of software testing and debugging until after hardware prototypes are built.

We have designed and implemented a simulation environment oriented to the needs of the developer of custom multi-DSP systems. Our simulator, Frigg, builds on the capabilities of a general-purpose behavioral simulator and of manufacturer-supplied processor simulators. Frigg allows a user to simultaneously view the detailed behavior of the hardware and software comprising a complete multiprocessor system. Frigg is currently being used to simulate a variety of multi-DSP systems, and has proven useful for verifying hardware designs, for performing detailed performance evaluation, and for testing software and hardware tools.

1. INTRODUCTION

1.1. The Need for System Simulation Tools

To satisfy the demanding requirements of real-time digital signal processing applications, a number of programmable digital signal processors (DSPs) have been developed. These chips are high-performance microprocessors, incorporating architectural features designed to enhance their effectiveness in signal processing applications. Quite often, system designers develop custom multiprocessor architectures using these processors. Developing hardware and software for such multiprocessor systems is a complex task, and there are few effective tools to aid the developer.

Instruction set simulators exist for most programmable DSPs, and behavioral-level simulators are available for some. However, previously there has been no readily available way for developers to accurately simulate the interactions between a programmable DSP and other digital hardware, or between multiple DSPs, while simultaneously modeling the effects of these interactions on software, and vice-versa.

This lack of flexible simulation capabilities has often meant that in practice, hardware designs for multiprocessor DSP systems are not simulated; they are tested and debugged only after prototypes have been built. To achieve maximum performance, software for DSP-based systems is generally written in assembly language. Without a simulation of the target architecture, detailed testing and debugging of this software must wait until debugged hardware prototypes are available. These factors lengthen development cycles and increase development costs.

Further, when developing multiprocessor systems for a specific application domain, a designer may wish to explore a range of architectural approaches. To evaluate the tradeoffs of each approach, ideally the designer would like to be able to execute programs (e.g., benchmarks) on the candidate architectures. This is especially important for applications with hard real-time requirements. Without a powerful simulation environment, such experimentation must be severely limited in scope and/or accuracy.

By providing a flexible, extensible environment for simulating DSP-based hardware and software designs, it is hoped that these problems can be overcome. Using Frigg, the system architect can rapidly set up simulations of candidate architectures, and thus may explore a wide range of possible system configurations without building prototypes or custom simulation programs. Once an architecture has been selected, the simulation may be used to allow full-scale testing and debugging of software, which can proceed simultaneously with the design of low-level hardware implementations. By allowing more informed design tradeoffs, and overlapping development of detailed hardware and software designs, it is hoped that Frigg will enable high quality designs to be realized faster than would otherwise be possible.

Frigg is a behavioral simulation environment. In behavioral simulation, the visible and relevant behavior of a device is simulated as accurately as necessary [1]. No reference need be made to the internal structure of a simulated device. However, low-level structural information may be used in the construction of a model for a device, if desired. By constructing a behavioral simulation model using a high-level language, it is possible to achieve performance that is dramatically better than that of gate- or switch-level simulations.

1.2. Frigg Strategy: A Novel Approach

Frigg is the first system known to the authors to interface a manufacturer-supplied VLSI processor simulator with a general-purpose behavioral simulator. By integrating existing simulations of DSPs, the difficult and time-consuming task of developing accurate behavioral models for these chips is avoided.

In implementing Frigg, we have illustrated several techniques for interfacing dissimilar simulation programs, and we have demonstrated that a useful simulation environment can be created by combining the capabilities of independently created software systems.

1.3. Related Work

A number of individual researchers and companies have proposed techniques for developing simulations of systems containing VLSI processing elements [2],[3],[4],[5]. However, few of these systems provide unified hardware and software views into the simulated system. Those that do provide such a facility generally rely on the user to write simulations for the processing elements and/or all other system hardware, usually without benefit of a simulation-oriented language and tools.

Manufacturers of VLSI processors increasingly use behavioral simulations early in the design process to explore architectural tradeoffs, and
later, to verify the correctness of an implementation. As these techniques become more prevalent, behavioral models of processors should become more widely available. Models supplied by the device manufacturer, which are products of the chip design process, should be very accurate, and should be available well before the device itself has been successfully manufactured. Frigg provides a mechanism for interfacing these models with a general-purpose behavioral simulation system.

2. IMPLEMENTATION

To create a flexible, extensible simulation environment, Frigg relies on the capabilities of Unix, the Thor simulator, X Windows, and manufacturer-supplied DSP simulators.

2.1. Thor

Frigg uses the Thor simulator from Stanford University [6] as the simulation substrate. Thor, a general-purpose, event-driven behavioral simulation system, is used to simulate all hardware elements other than DSPs, and to simulate all interconnections between elements. Frigg integrates the capabilities of Thor with those of manufacturer-supplied DSP simulators. Thor behavioral models of hardware devices are written in CSL, a hardware description language that is based on the "C" programming language. Models are preprocessed by Thor, and then compiled by the standard Unix "C" compiler. Thus, Thor hardware models can contain standard "C" program code. This allows the possibility of interfacing other simulation programs with Thor, either directly, through "C" language function calls, or indirectly, through other mechanisms like the Unix inter-process communications (IPC) facility. Either of these mechanisms can be used in Frigg. Thor's CSL netlist specification language is used to specify the connection of circuit elements.

For the most part, Frigg simulations are specified by the user just like any other Thor simulations. However, when processing elements are referenced, a special model provided by Frigg is invoked. This model appears to Thor to be an ordinary simulation model. In fact, the model does not implement a simulation of the processor, but rather contains the interface code necessary to effect communication with and control over other functions or processes which implement the processor simulation.

The Thor user interface provides a graphic logic analyzer utility. This window-based tool behaves much like a real logic analyzer. The user may connect analyzer probes to signals in the simulated system, and observe the behavior of the system as waveforms in the analyzer window.

2.2. DSP Simulator

The current implementation of Frigg supports the Motorola DSPS56000 family of programmable digital signal processors. Support was implemented for this processor first because we are currently working with this processor in our lab, and because certain characteristics of the Motorola simulator make it particularly suitable for interfacing with other simulators. However, Frigg has been developed with the intention of interfacing a variety of processor simulators.

Sim56000, Motorola's simulator for the DSP56000 processors, is a complete, stand-alone simulator for the DSP [7]. Sim56000 provides an interactive user interface which is oriented towards development and debugging of program code. In these respects, Sim56000 is very similar to simulators provided by many other DSP manufacturers for their processors. These simulators generally provide little or no support for simulating multiprocessor systems, or for simulating complex hardware external to the DSP.

Sim56000 has two important differences from most other processor simulators, though. First, Sim56000 is a complete behavioral simulation of the processor, not just an instruction set simulator. This means that Sim56000 accurately models the behavior of each of the processor's signal pins. Second, Sim56000 is provided to the user in the form of an object-code library of user-callable "C" language functions. This allows the user to write a custom "C" program to control the execution of the DSP simulation. The combination of these two features has allowed us to interface Sim56000 with Thor, combining the best features of both systems.

2.3. Interconnection of Simulators

Frigg provides two mechanisms for connection between the general purpose simulator (Thor) and DSP simulators. The first method uses the inter-process communications (IPC) facilities of Unix. A less flexible but more efficient approach links the programs together and uses "C" language function calls for communication between simulators.

Unix IPC

The Unix IPC facilities provide a very flexible and powerful mechanism for connecting simulators. With this approach, the simulators are executed as separate processes. Thor acts as the "parent" and coordinating process. For each DSP element specified in the simulated system, a Sim56000 process is initiated. Each Sim56000 process is assigned to its own terminal emulator window under X Windows. Separate IPC connections are established between Thor and each of the Sim56000 processes. Additional windows are created for Thor's logic analyzer utility. This multi-window approach creates a convenient way for the user to simultaneously monitor and interact with different views of the simulated system. The user can simultaneously monitor the internal states of each of the processors, while observing the detailed behavior of other system hardware with the logic analyzer utility. Using IPC to connect independent cooperating simulation processes also allows for the possibility of running distributed simulations. Compared with the speed of moving data with conventional function calls, though, using IPC is significantly slower.

Function Calls

Since Motorola's Sim56000 simulator is provided in the form of a library of object-code "C" language functions, and since Thor allows essentially arbitrary "C" code to be included in the behavioral description of a hardware module, it is possible to link Sim56000 with the Thor simulation to create a single executable module. In this case, Thor and Sim56000 communicate through the use of conventional "C" language subroutines. This mechanism is much more efficient than using the Unix IPC facilities; however, it lacks the flexibility of the IPC scheme described above.

Specifically, this mechanism requires that the processing element simulator be provided in the form of a function library. In addition, it makes handling multiprocessor simulations and providing a multi-window interface more difficult.

Thor is an event-driven simulator, while Sim56000, like most processing element simulators, is time-driven. This means that some care is required when connecting the two. The current implementation of Frigg uses a very straightforward approach. The basic time step of the Sim56000 simulator is one-fourth of an instruction cycle; this is the resolution necessary to correctly simulate the processor's bus and I/O timing. Within a simulated system in Frigg, processors are assumed to be synchronous. A standard Thor library model is used to generate a periodic clock signal, which is connected to each processor model. In the case of the DPS56000 model, the clock signal frequency is twice the instruction cycle rate. Each time the clock signal changes its value, Thor will evaluate the processing element model. This ensures that the processing element simulation is advanced at least twice per clock cycle. Code within the Thor interface model which communicates with the processing element simulator ensures that the processing element simulation is advanced no more than twice per clock cycle.

Each time Thor "evaluates" the processing element model, the Thor model representing the processing element translates the logic values present at the processor's pins into values meaningful to Sim56000, transmits these values to Sim56000, and commands Sim56000 to advance the processor simulation by one time step. Thor then waits for Sim56000 to transmit back the new logic values on the processor pins, and then continues with the rest of the simulation. This scheme is simple to implement, but it creates a large amount of communication between simulators. This large amount of communication can become a limiting factor in the speed of the simulation, particularly when IPC is used to connect simulators.

Running a Frigg simulation is very simple, and involves the following steps.
First, the user must specify the system to be simulated. In general, this involves choosing appropriate hardware models from a library, possibly writing some new models using the CHDL language of Thor, and specifying the interconnection of hardware elements through the use of one or more CSL netlist files. The DSP56000 model is treated as an ordinary Thor model. Each DSP56000 in the simulated system may have an object code program and command file associated with it. The command file specifies a sequence of simulator commands for each DSP in the system. For example, the command file may instruct the simulator to load and execute a particular object code file, to set software breakpoints, or to periodically display certain information.

After specifying the system to be simulated, the user begins execution of the simulation by issuing a command that causes Thor to create and run an executable simulation module. During the simulation, the user can inspect any signals in the system using the Thor logic analyzer, and can individually monitor and interact with each DSP model. Using simple X Windows operations, the user can inspect, de-liconify, move, foreground or background a particular processing element or Thor window. To interact with a particular component, the user simply selects the appropriate window with the mouse, and then uses the command line interface of Thor or the processing element simulator to execute operations. This allows the simultaneous monitoring of the detailed behavior of system software, within the processors, and hardware, outside of the processors.

In addition to (or instead of) monitoring the simulation interactively while it executes, the user can use the file output capabilities of Thor and Sim56000 to record virtually any kind of information from the simulation in disk files. When the simulation terminates, these files can be viewed with a text editor or a graphing program, or analyzed with other software. For example, the output samples of a DSP system can be recorded in a file, and then plotted or analyzed with the use of signal analysis software.

3. RESULTS

3.1. Example

Figure 1 illustrates Frigg in use. In the figure, Frigg is simulating a custom four-processor architecture based on the DSP56000. Four processor simulator windows are visible on the right-hand side. On the left, a Thor logic analyzer window monitors the memory buses and control lines of each of the four processors. In this example, the analyzer window is being used to observe the behavior of the bus arbitration hardware, and to ensure that all of the processors obey system's bus access protocol.

3.2. Performance

The performance of Frigg simulations is a factor which may seriously limit their applicability beyond DSP systems. For simulated systems with a small number of processors (e.g., 1-10) and a moderate amount of external simulated hardware (e.g., some large memories, buses, bus arbitrators, and peripheral interfaces), Frigg simulations execute at a speed of approximately 60 processor-instructions per second, if the 'C' function call interface is used for communication between processors. Performance degrades to about 15 processor-instructions per second if the IPC interface is used. These speeds were measured on a Sun Microsystems 3/60 workstation with 12MB of RAM. Simulations at these speeds have proven useful for testing software for digital signal processing algorithms which contain a minimal amount of decision making; however, they would not be adequate for testing complex software, for example operating systems. Future work on Frigg will explore speeding up simulations through the use of distributed processing and other means.

3.3. Applications

While Frigg simulations are probably not useful for testing large and complex software systems, they have already proven useful in a number of applications. Frigg simulations provide a combined hardware and software view of the simulated system. The DSP device simulators can be used to examine the detailed behavior of programs, allowing, for example, breakpoints, single stepping and tracing. At the same time, the Thor simulation of the external hardware allows detailed monitoring and logging of the activity of any buses or signals in the system.

3.4. Performance Evaluation of DSP Systems

Many digital signal processing applications have very stringent hard real-time constraints. For example, a given hardware and software system may be required to execute a particular algorithm at a specified sample rate and degree of precision. Thus, it can be crucial that system designers have the ability to evaluate the performance of their hardware and software before actually committing to a design. With conventional tools, it has not been possible to obtain precise system performance information, except for the simplest of uniprocessor configurations.

Frigg simulations have been used to determine the precise number of processor cycles required for a given hardware implementation to execute a particular algorithm or code fragment. Such simulations can be very useful for informing the engineering tradeoffs that must be made in designing the architecture and implementation of a complex system. Since Frigg simulations can be specified, compiled, and run very quickly, it may be faster to obtain performance information with Frigg than from manual calculations.

3.5. Development and Testing of Hardware Designs

Conventionally, developers of programmable DSP-based systems have not had tools to allow them to simulate their board- and system-level designs. This has meant that design errors are often found only after prototypes have been fabricated. Also, when prototypes become available, there may be uncertainty as to whether a malfunction is due to a design error or to an error in the fabrication of the prototype. Once prototypes have been fabricated, it can be costly and time-consuming to correct design mistakes. Ideally, the designer should be able to test her design using simulation, so that when the first prototype is fabricated, she has a high level of confidence that it will work. Such techniques have proven extremely effective for the development of single-chip VLSI systems. Frigg brings powerful design verification capabilities to the realm of DSP system design. Using Frigg simulations, we have developed diagnostic routines for a target system. The results of tests on a simulated system can later be compared with the results of physical testing of the fabricated system.

3.6. Testing of Software

In designing any computer system, it is often desirable to be able to thoroughly test and debug software without having to wait until the hardware implementation has been completely designed, prototyped, and debugged. In practice, particularly for DSP-based systems, this ideal has been difficult to achieve. The Frigg simulation environment allows software for complex multiprocessor DSP-based systems to be realistically tested and debugged. When software is tested under Frigg, interactions between processors and the effects on software of circuits outside of the DSP itself are accurately modeled.

Further, since Thor simulations of subsystems are behavioral in nature, it is possible to model the functionality of complex system components using high-level descriptions. This means that functionally-correct system simulations can be performed even before details of the hardware implementation have been resolved. Also, software "test instruments" can be quickly created, and then used to facilitate thorough testing of the simulated system.

In our DSP lab, we have used Frigg simulations to test the correctness and performance of software. Because instrumenting complex multiprocessor hardware can be difficult, and because software debugging tools for such systems are virtually non-existent, Frigg simulations often prove to be more useful than the hardware itself for diagnosing software problems and measuring performance.

A related research project underway at U.C. Berkeley is concerned with developing high-level software tools for multiprocessor DSP systems. The system under development, Gabriel, is designed to be extensible across a wide range of multiprocessor architectures (8). Frigg is being used with Gabriel to provide simulations of a wide variety of target architectures, which allows more thorough testing of Gabriel's capabilities than would otherwise be possible.
4. CONCLUSION

Multiprocessor systems based on programmable digital signal processors offer a promising approach for tackling a variety of applications with stringent real-time requirements and demanding computation needs. The current dearth of tools supporting the design and development of such systems creates a serious impediment to their use. In implementing Frigg, we have tried to illustrate how useful tools for multiprocessor system development can be built. Further, we hope to encourage a range of manufacturers of microprocessors and other VLSI devices to make behavioral simulations for their chips available to users in a form which allows them to be interfaced with more general simulation environments.

REFERENCES


Figure 1. An illustration of Frigg simulating a custom four-processor architecture based on the DSP56000. In the Thor logic analyzer window (left side) the memory buses and control lines of each processor are being monitored. The analyzer window is being used to observe the behavior of the bus arbitration hardware, and to ensure that all of the processors obey the software bus access protocol. On the right are visible four DSP simulator windows, one for each processing element in the simulated system. The window for processor 0 displays the state of the processor registers, and indicates that this processor has halted the simulation upon encountering a breakpoint. The window for processor 2 shows a disassembled section of the processor's program code.