[33] E. A. Lee and J. Bier, “Architectures For Statically Scheduled Dataflow,” *Journal on Parallel and Distributed Systems*, December 1990.

[34] G. Sih and E.A. Lee, “Dynamic-Level Scheduling for Heterogeneous Processor Networks,” *Second IEEE Symposium on Parallel and Distributed Processing*, December 1990.

[35] E. A. Lee and S. Ha, “Scheduling Strategies for Multiprocessor DSP”, *Proc. of GLOBECOM*, Dallas, Texas, November, 1989.

[36] J. Bier and E. A. Lee, “Frigg: A Simulation Environment for Multiprocessor DSP System Development”, *Proc. of Int. Conf. on Computer Design*, Boston, MA, October, 1989.

[37] E. A. Lee, W.-H. Ho, E. Goei, J. Bier, and S. Bhattacharyya, “Gabriel: A Design Environment for DSP”, *IEEE Trans. on ASSP*, November, 1989.

[38] E. A. Lee, “Recurrences, Iteration, and Conditionals in Statically Scheduled Block Diagram Languages”, in *VLSI Signal Processing III*, Ed. R. W. Brodersen and H. S. Moscovitz, IEEE Press, New York, 1988.

[39] W.-H. Ho, E. A. Lee, and D. G. Messerschmitt, “High Level Data Flow Programming for Digital Signal Processing”, in *VLSI Signal Processing III*, Ed. R. W. Brodersen and H. S. Moscovitz, IEEE Press, New York, 1988.

[40] E. A. Lee and D. G. Messerschmitt, “Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing” *IEEE Transactions on Computers*, January, 1987.

[41] E. A. Lee and D. G. Messerschmitt, “Synchronous Data Flow” *IEEE Proceedings*, September, 1987.

[16] J. Buck, S. Ha, E.A. Lee, D.G. Messerschmitt, "Ptolemy: A mixed Paradigm Simulation/ Prototyping Platform", *Proc. of Speech Tech 1991*, New York, NY, April 23-25, 1991.

[17] S. Bhattacharyya, "Scheduling Synchronous Dataflow Graphs for Efficient Iteration", Master's Thesis, EECS Dept., Univ. of Calif., Berkeley, May, 1991.

[18] S. Ha, E. A. Lee, "Quasi-Static Scheduling for Multiprocessor DSP", *Proc. of ISCAS*, Singapore, June 1991.

[19] J. Buck, S. Ha, E. A. Lee, and D.G. Messerschmitt, "Ptolemy: A Platform for Heterogeneous Simulation and Prototyping, *Proc. 1991 European Simulation Conference*, Copenhagen, Denmark, June 17-19, 1991.

[20] E. A. Lee, "Consistency in Dataflow Graphs", *Proceedings of the International Conference on Application Specific Array Processors*, (Barcelona, Spain), IEEE Computer Society Press, Los Alamitos, California, September 1991.

[21] Soonhoi Ha and E.A. Lee, "Compile-Time Scheduling and Assignment of Dataflow Program Graphs with Data-Dependent Iteration," *IEEE Transactions on Computers*, November, 1991.

[22] J. Buck, S. Ha, E. A. Lee, D. G. Messerschmitt, "Ptolemy: A Mixed-Paradigm Simulation/Prototyping Platform in C++", *Proc. C++ At Work Conference,* Santa Clara, CA, November, 1991.

[23] P. D. Lapsley, "Host Interface and Debugging of Dataflow DSP Systems", MS Thesis, Electronics Research Laboratory, University of California, Berkeley, CA 94720, December, 1991.

[24] A. Kalavade, "Hardware/Software Codesign Using Ptolemy", MS Report, Electronics Research Laboratory, University of California, Berkeley, CA 94720, December, 1991.

[25] G. Sih and E. A. Lee, "List Scheduling Modifications to Account for Interprocessor Communication Within Interconnection-Constrained Heterogeneous Processor Networks", *Proceedings of the Int. Conf. on Parallel Processing*, February, 1990.

[26] J. Bier and E. A. Lee, "A Class of Multiprocessor Architectures for Real-Time DSP," *Proceedings of ISCAS 90*, New Orleans, May, 1990.

[27] S. How, "Code Generation for Multirate DSP Systems in Gabriel," MS Report, ERL, EECS Dept., UC Berkeley, CA 94720, May, 1990.

[28] M. Grimwood, "An Application Using Gabriel: Design and Implementation of a Free-Space Digital Infrared Communication Link," MS Report ERL, EECS Dept., UC Berkeley, CA 94720, August, 1990.

[29] M. Fratt, "Speech Processing Using The Gabriel DSP System," MS Report ERL, EECS Dept., UC Berkeley, CA 94720, August, 1990.

[30] M. P. O'Reilly, "The Design of a 16-QAM Passband Data Modem Using Gabriel, Plan II," MS Report, ERL, EECS Dept., UC Berkeley, CA 94720, August, 1990.

[31] J. Bier, E. Goei, W. Ho, P. Lapsley, M. O'Reilly, G. Sih and E.A. Lee, "Gabriel: A Design Environment for DSP," *IEEE Micro Magazine*, October 1990, Vol. 10, No. 5, pp. 28-45.

[32] J. Bier, S. Sriram and E.A. Lee, "A Class of Multiprocessor Architectures for Real-Time DSP," *VLSI DSP IV*, ed. H. Moscovitz, IEEE Press, November, 1990.

F. Bibliography of Publications Relating to Ptolemy

[1] E. A. Lee, "Multidimensional Streams Rooted in Dataflow", EECS Dept., UC Berkeley, August 1, 1992. submitted to *IFIP Working Conference on Architectures and Compilation Techniques for Fine and Medium-Grain Parallelism*, Orlando, FL, January, 1993.

[2] J. Buck, S. Ha, E. A. Lee, D. G. Messerschmitt, "Ptolemy: a Framework for Simulating and Prototyping Heterogeneous Systems", to appear in *International Journal of Computer Simulation*, special issue on "Simulation Software Development," 1992.

[3] J. Pino, S. Ha, E. Lee, J. Buck, "Software Synthesis for DSP Using Ptolemy", invited paper in the *Journal on VLSI Signal Processing*, special issue on "Synthesis for DSP", to appear.

[4] S. Bhattacharyya and E. A. Lee, "Scheduling Synchronous Dataflow Graphs for Efficient Looping," to appear in *J. of VLSI Signal Processing,* 1992.

[5] Gregory Walter, *ATM, Speech Coding, and Cell Recovery*, MS Report, December, 1992.

[6] J. Buck and E. A. Lee, "The Token Flow Model," presented at *Data Flow Workshop*, Hamilton Island, Australia, May, 1992.

[7] A. Kalavade and E. A. Lee, "Hardware/Software Co-design Using Ptolemy — A Case Study," *Proc. of the IFIP Int. Workshop on Hardware/Software Co-design*, Grassau, Germany, May 19-21, 1992.

[8] S. Ha, "Compile-Time Scheduling of Dataflow Program Graphs with Dynamic Constructs," Ph.D. Dissertation, EECS Dept., University of California, Berkeley, CA 94720, April 1992.

[9] E. A. Lee, "A Design Lab for Statistical Signal Processing," *Proceedings of ICASSP,* San Francisco, March, 1992.

[10] G.C. Sih, E.A. Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures", to appear, *IEEE Trans. on Parallel and Distributed Systems,* 1992.

[11] E. A. Lee, "Static Scheduling of Data-Flow Programs for DSP," in *Advanced Topics in Data-Flow Computing,* ed. J.-L. Gaudiot and L. Bic, Prentice-Hall, 1991.

[12] E. A. Lee and J. C. Bier, "Architectures for Statically Scheduled Dataflow", reprinted in *Parallel Algorithms and Architectures for DSP Applications,* ed. M. A. Bayoumi, Kluwer Academic Pub., 1991.

[13] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Multirate Signal Processing in Ptolemy", *Proc. of the Int. Conf. on Acoustics, Speech, and Signal Processing*, Toronto, Canada, April, 1991.

[14] E. A. Lee, "Consistency in Dataflow Graphs", *IEEE Transactions on Parallel and Distributed Systems",* Vol. 2, No. 2, April 1991.

[15] Gilbert C. Sih, "Multiprocessor Scheduling to Account for Interprocessor Communication", Ph.D. Thesis, ERL, UC Berkeley, CA 94720, April 22, 1991.

E.2 Wireless Networking

The wireless work is collaborative with Profs. Kahn and Linnartz, and is attempting to achieve high speed wireless access to the network. We are using Ptolemy as a basis for the fading channel, modulation, PLL simulations. An important capability of Ptolemy is to model at the timing and circuit simulation levels, so we can study interactions between circuit designs (preamps etc.) and other parts of the system.

E.3 Cell-Relay Networks

To demonstrate a large heterogeneous simulation, we have constructed a full cell-relay (ATM) network. While it is simplified in some respects, it includes the three key elements: cell-relay transport (DE domain), call processing (MQ domain), and video codec (SDF domain). CPE's set up random calls through the network, which are routed by the call processing, and signal processing based CPE's transmit packets through the network. The 2x2 switching elements in the Batcher-Banyon switches are modeled in SDF, including their routing tables, and the interface of these routing tables to the MQ call processing was an interesting challenge.

Currently another project is focusing on the buffering and associated control at the interface between the synchronous signal processing world and the asynchronous transport: the cell assembly operation. As a design exercise, we are carrying this design down to the board level, utilizing the DE and Thor domains of Ptolemy. What is interesting here is the ability to model the interaction of this board with the network (congestion control) and signal processing (flow control) in the context of the hardware model.

E.4 Video Signal Processing

We are developing models in Ptolemy for standard video signal processing algorithms (DCT etc.). Later this will be merged with the networking simulations (cell relay) to study issues in the interaction between transport and signal processing for multipoint and interactive video services. For example, the compression must be designed taking into account the cell-loss mechanisms, and also must accept flow-control directives from the network.

E.5 Power Networks

Prof. Varaiya and Wu at Berkeley are using Ptolemy for the modeling of the communication and control infrastructure of an electric power network. Theirs is actually the first large-scale communication system modeling effort in Ptolemy. Subsequently they plan to model the power grid itself, and then connect these two heterogenous large-scale systems.

run any application. Instead of functional blocks, it has a suite of generic blocks, each with a parameter that specifies its execution time. Furthermore, the target is parameterized so that we can modify the interprocessor communication times and test the efficacy of the scheduler. The value of each parameter used to generate figure 12 is given below:

- Runtimes of the blocks: ranging from one time unit to five time units.
- Time to send one data packet: one time unit.
- Time to receive one data packet: one time unit.
- Communication interferes with processing (no separate IPC unit).
- Number of processors: 4
- Processors contend for communication resources.

A Gantt chart is shown below the block diagram. A vertical line in the Gantt chart causes the blocks that fire at that time to be highlighted. Note that there is only sufficient concurrency in this test program to use three of the four processors, so the scheduler does not make use of the last processor.

All of the parameters can be modified, and the scheduler will adapt. For instance, if the communication time is increased from one time unit to 10, say, we find that the scheduler maps the entire application onto a single processor. Communication is too expensive to justify parallelizing such a simple application.

The first two schedulers support only the SDF model of computation. A third scheduler has been developed that supports the DDF model of computation. All of these schedulers will be compatible with any code generation domain compatible with these models of computation. Hence, they can be used to generate parallel programs in any language.

E. Representative Applications

Ptolemy is currently be used for a number of applications, at Berkeley and elsewhere. We will briefly describe the current activity at Berkeley.

E.1 Signal Processing

A wide variety of signal processing applications have been developed using Ptolemy, including several adaptive filtering applications, power spectrum estimation, several parametric and waveform coding techniques based on linear prediction, communication channel modeling, digital communication receivers, beamforming, digital filter design, chaos simulations, phase-locked loops, image coding, and music synthesis. Many of these applications are distributed as with the Ptolemy code as demonstrations. Ptolemy is also being used to teach both graduate statistical signal processing and undergraduate digital signal processing.

D.2 The role of graphics in parallel programming

Compilation of parallel programs and programmer visualization of parallelism both demand the same missing ingredient: elegant representations of programs that are compatible with parallel computing. This need not mean new languages, but it almost certainly means new program structuring. Our experience indicates that combinations of concise, textual representations of arithmetic and sequential operations with graphical representations of higher-level program structure can be effective, at least in the domain of signal processing.

D.3 Current Status of Parallel Programming Using Ptolemy

Three parallel schedulers have been built into two generic code generation domains in Ptolemy. By “generic domains” we mean domains that form a base class for specific code generation domains. Hence, the schedulers can be used for a variety of targets without any changes. An example program in such a domain is shown in figure 12. This domain consists only of base classes to be used in actual code generation domains, and hence does not actually

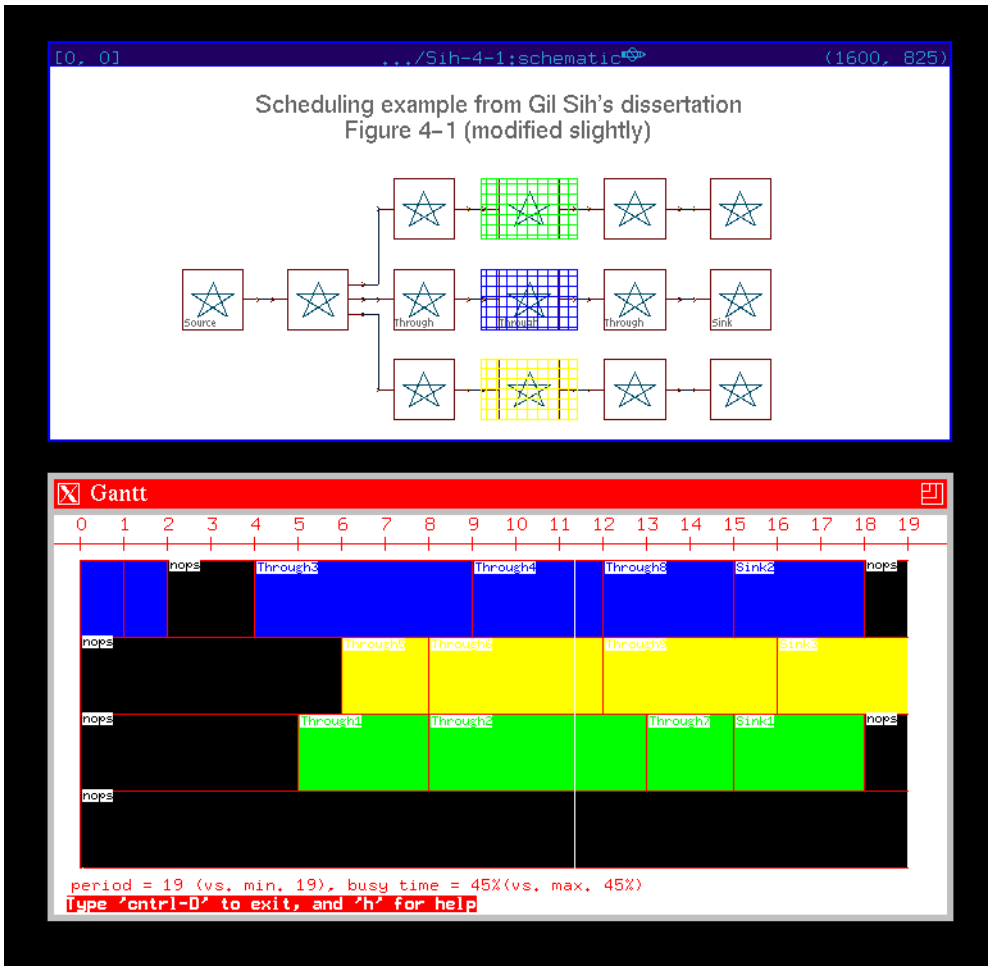

FIGURE 12. A program in a generic code generation domain, and a parallel schedule generated for that program.

A practical application for the latter capability is the design of a multimedia workstation, where programmable hardware (FPGAs) might be used to interface to a variety of audio and display devices, one or more programmable DSPs might be used to manage encoding and decoding of real-time signals, a processor running a real-time O/S might handle all real-time control, and a processor running Unix might manage the interaction with the user. Ptolemy would unify this highly heterogeneous environment.

D. Parallel Programming using Ptolemy

Parallel programs are not so much "written" as "designed". Writing is sequential, left-to-right, top-to-bottom, while design builds relationships between parts. Even with ordinary structured and object-oriented programming, the writing metaphor is weak. It requires augmentation with specialized text editors and hypertext facilities such as class browsers and tag managers. But the breakdown of the metaphor is complete with parallel programming.

A "design" makes a whole from parts without any implication of sequentialism. In parallel programs, the parts may be processing elements or concurrent processes, but far better, they should be parts of the application being designed. The interaction between these parts, and between these parts and a user, is after all what concerns the designer (the "programmer").

D.1 Requirements of parallel programming

Parallel programming requires visualizing the parallelism in an application. This need not be any more unnatural than visualizing parallelism in, say, a circuit schematic. Half-hearted enhancements to traditional languages, like "for-all" constructs, hardly satisfy the need, since they only hint to a compiler of some inner-loop parallelism that might be exploited. Communicating sequential processes, such as those supported by the Occam language, in principle allow more flexibility, but they leave too much of the scheduling work to the programmer. Some attempts to automatically exploit parallelism have met limited success either because they started with sequential languages, or because they were too specialized. Specialized techniques, such as automated mapping onto systolic arrays, can be effective for specific algorithms, but few if any applications consist entirely of a single algorithm. To be useful, these techniques have to be combined with others into an environment that achieves generality through a suite of specialized methods.

Parallel programming also requires fundamentally new compiler and debugger technology where the focus is less on code generation and more on scheduling. By "scheduling", we mean partitioning, mapping, ordering, and timing of tasks on processors. It also includes routing of communications. The compiler should assume responsibility for as many of these as possible, but with hooks for the programmer to tune the schedule if desired. Dataflow machines have shown that it is not reasonable in most cases for run-time hardware to assume such responsibility. And the limited penetration of today's commercial parallel machines has shown that programmers are not willing to assume such responsibility themselves.

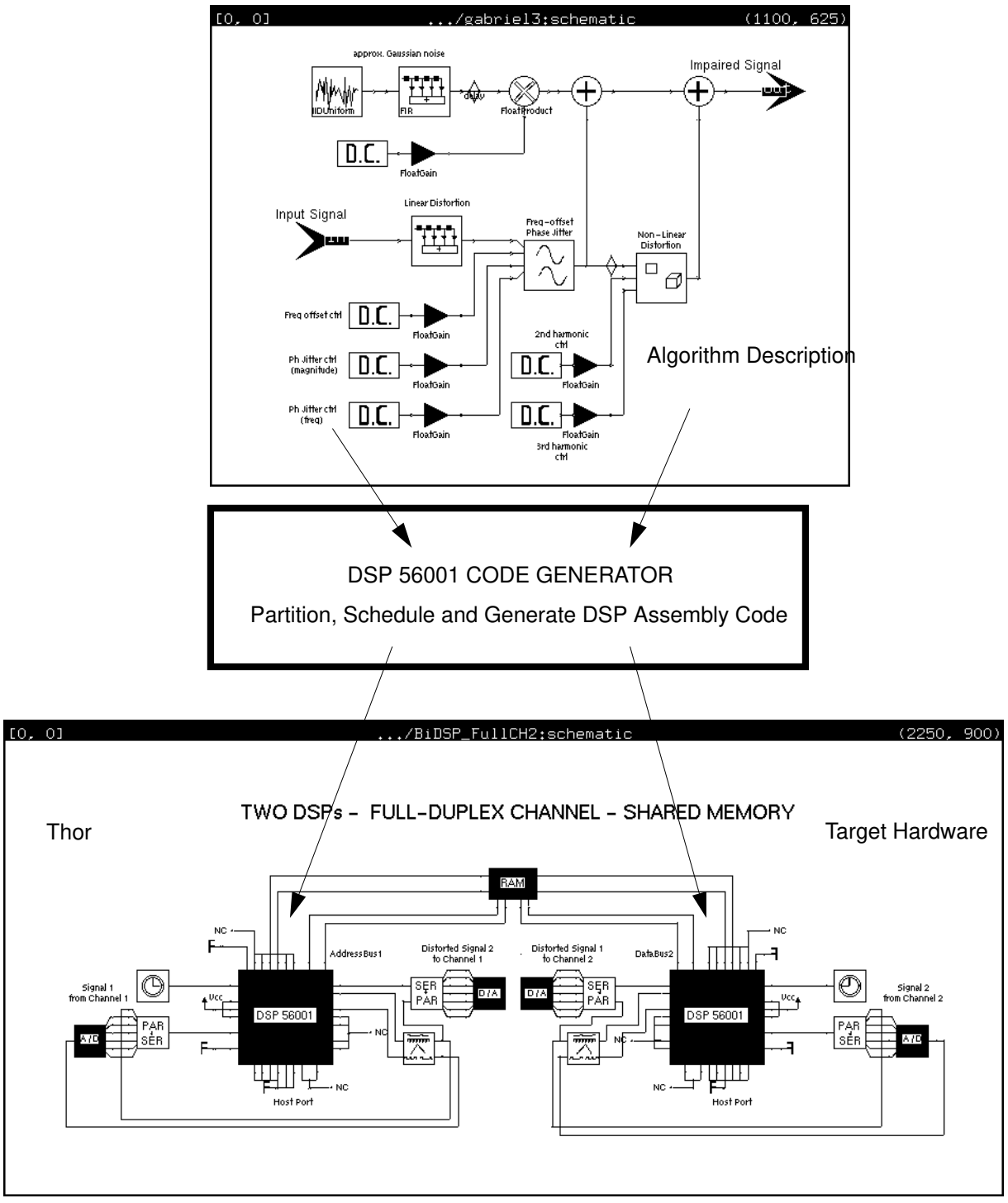

FIGURE 11. A hardware design (bottom) containing programmable DSPs can be developed together with the software (top) that will run on the DSPs. This figure shows the top level only of a telephone channel simulation algorithm (top window) being mapped onto a board design with two Motorola DSP56001 DSPs.

are needed by the destination equipment. Above this network model is a highly simplified signal processing system making use of the network. A sinusoid is generated and packetized, one sample per packet, and launched into the network. At the receiving end, packets are used to reconstruct the sinusoid at the same sample rate. The packets are used in the order of arrival, so the samples of the sinusoid get randomly scrambled, as shown in the upper plot. Real-time constraints are modeled, so if packets do not arrive in time, earlier packets are re-used.

In practical applications with similar structure, both the network model and the signal processing will be much more elaborate. Ptolemy is currently being used to evaluate video encoding algorithms for transmission over ATM (asynchronous transfer mode) networks, as shown in figure 6.

C.2 Hardware/Software Codesign

Most electronic systems mix custom circuit designs with programmable commodity parts. Ptolemy supports such designs as a unit, since using the various domains described above, all parts of the system can be modeled. For example, the Thor domain can be combined with a code generation domain to design boards that mix custom hardware with programmable DSPs. Since the hardware and software are both modeled within the same software framework, a designer can easily explore tradeoffs between hardware and software implementations of various functions.

An example of such a design is shown in figure 11. The top window shows the top level only of an algorithm that simulates the impairments of a telephone channel. This algorithm is fairly complicated, including linear and non-linear distortion, frequency offset, phase jitter, and additive Gaussian noise. The design is built in a code generation domain compatible with the SDF model of computation. The bottom window shows a hardware design containing two programmable DSPs communicating through a dual ported shared memory. This hardware might be used to implement the telephone channel simulator for production-line testing of voiceband data modems.

The design shown in figure 11 is one of many that could accomplish the stated objectives. Using Ptolemy, the entire design, ranging from algorithm development to circuit design, can be carried out within a unified environment. This enables exploration of many design alternatives before resources are committed to hardware prototyping.

C.3 Multi-Platform Implementations[1]

The interface between domains in Ptolemy is being generalized to permit different parts of a Ptolemy system to run on distinct hardware platforms. A simple form of this distributes Ptolemy simulations among multiple Unix workstations. A more elaborate form spawns stand-alone C++ processes that execute on remote platforms, communicating with the Ptolemy process through a network, although this has not yet been demonstrated. A still more elaborate form generates code for non-Unix machines, in C or assembly language, downloads the code, and manages the communication between system components transparently.

1. This section describes work in progress, not capabilities in the version of Ptolemy currently being distributed.

ting data, speech, images, and video, requires consideration of all these issues. For example, to carry real-time speech, an encoding algorithm that tolerates lost or delayed packets may be required. The encoder may also be required to monitor the network for congestion, and adapt its compression ratio accordingly. A video transmission network may need to perform video compositing within the network, rather than just at the terminals.

Such multimedia designs are fundamentally heterogenous. Network design, which is naturally done with the DE domain in Ptolemy, must be combined with signal processing design, which is most naturally accomplished in the SDF or DDF domains.

A simplified example that combines the SDF and DE domains is shown in figure 10. In this example, the lower diagram models a highly simplified packet-switched communication network, in which packets randomly traverse one of two paths. The upper path has no delay. The lower path has random delay. At the receiving end, a queue stores incoming packets until they

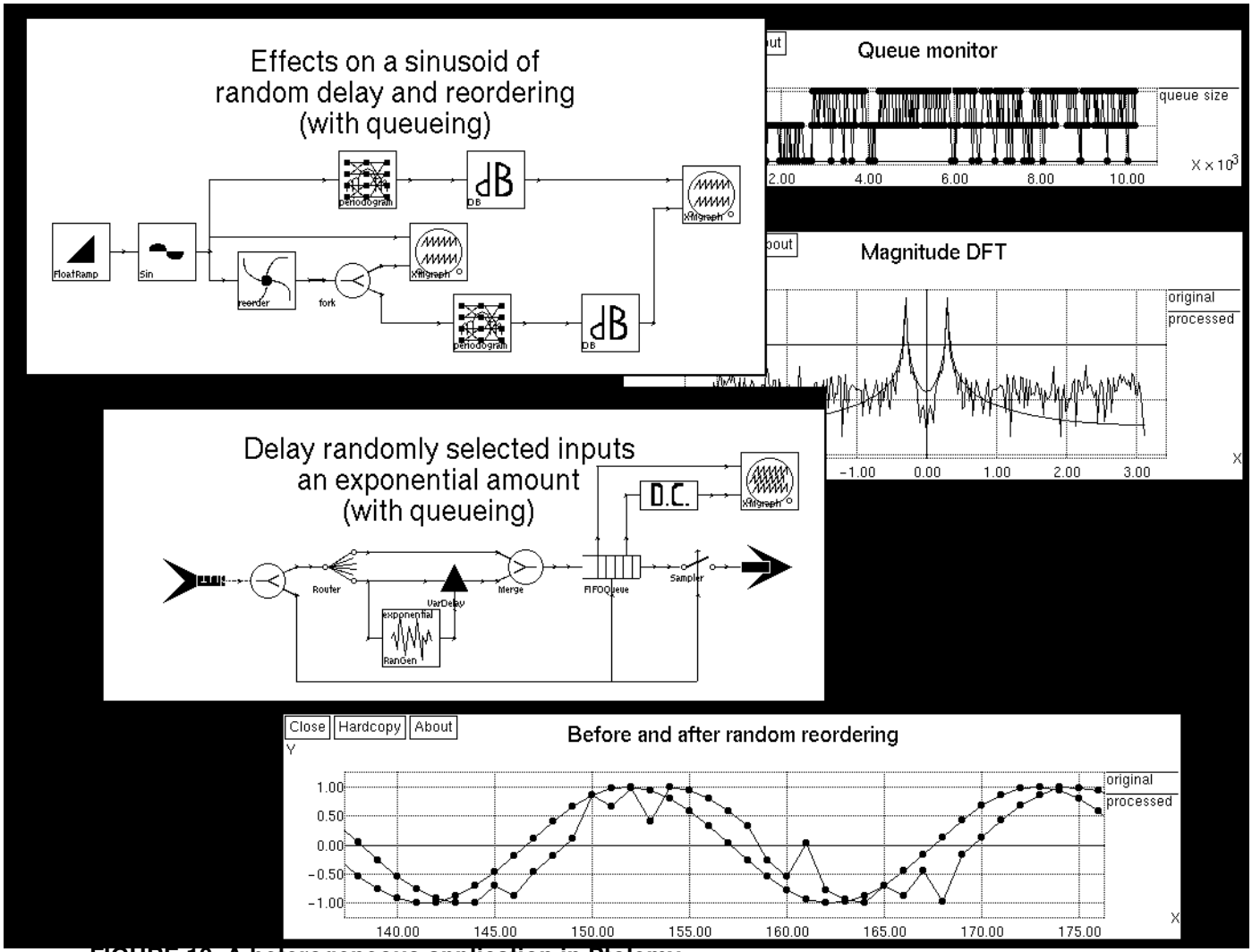

FIGURE 10. A heterogeneous application in Ptolemy.