Electrical Engineering and
Computer Sciences

# Hardware / Software Co-Design Using Ptolemy - A Case Study

Asawaree Kalavade
Edward A. Lee

University of California at Berkeley,
Berkeley, CA 94720.

## Abstract

Ptolemy is an environment for simulation and prototyping of heterogeneous systems. By supporting the co-existence and interaction of different models of computation, Ptolemy facilitates mixed-mode system simulation, specification, and design as well as generation of DSP assembly code from a block diagram description of the algorithm. These features render Ptolemy suitable for hardware/software codesign. This case study demonstrates the use of Ptolemy for hardware/software co-design. The test case is a telephone channel simulator that generates EIA-specified channel impairments for voice-band data modem testing - where the hardware comprises custom hardware coupled to programmable DSP chips, and the software is the code running on these programmable processors. The codesign methodology using Ptolemy is illustrated via the development and evaluation of a sequence of designs for this telephone channel simulator. These designs address multiprocessor communication, scheduling and code partitioning issues, as well as issues of system-level hardware/software partitioning of functionality.

*Appeared In: Proceedings of the First Intl. Workshop on Hardware/Software Codesign, Colorado, Sept. 1992.*

# 1.0 Introduction

## 1.1 What is Co-Design?

In a traditional design strategy, the hardware and software partitioning decisions are fixed at an early stage in the development cycle and the hardware and software designs are developed separately from then onwards. With advancements in technology, however, it becomes possible to obtain special-purpose hardware components (ASICs) at a reasonable cost and development time. Some designs also call for some programmability in the end product. This suggests a more flexible design strategy, where the hardware and the software designs proceed in parallel, with feedback and interaction between the two as the design progresses. The final hardware/software split can then be made after the evaluation of alternative structures with respect to performance, programmability, non-recurring (development) costs, recurring (manufacturing) costs, reliability, maintenance, and evolution of design. This design philosophy, which helps reduce the time to market, is called hardware/software codesign.

## 1.2 Scope of Co-Design Strategies

Codesign strategies can be applied to designs at different levels:

1. **Processor Design**: An optimized application-specific processor can be achieved by jointly designing the instruction set and the program for the application. This is a very difficult design problem.

2. **System-level Design:** Codesign methodology is applicable at the system level, where an algorithm must be partitioned between custom hardware, and software running on commodity programmable components. The hardware would typically include discrete components, ASICs, DSP cores, microprocessors, microsequencers, microcontrollers, or semi-custom logic developed using FPGAs or logic synthesis tools. There are many possibilities for partitioning a given design between hardware and software components. Evaluation of these alternatives using system-level simulation of the hardware and software is a key aspect of codesign.

3. **Application-Specific Multiprocessor System Design**: Design of an application-specific multiprocessor system is challenging because it involves selection of the appropriate number of processors, the inter-processor-communication (IPC) strategy, and the design of the software for the application. Software synthesis requires partitioning and scheduling the code over the processors, so scheduling techniques must be capable of adapting to changing hardware configurations. Thus, design of such an application-specific multiprocessor system is an iterative

process — it involves trade-offs associated with the selection of the optimal hardware structure and software partitioning.

With the help of a case study, this work presents a study of a codesign methodology at the second and third levels. We do not address processor design, but instead assume that the programmable components are commodity parts.

### 1.3 Co-Design for DSP Applications

In this study, we restrict the scope of codesign to digital signal processing applications. DSP applications have the desirable feature that their implementation is simple, yet demands high performance and throughput. A variety of commercial DSP microprocessors can be used for most of the sophisticated signal processing required in these applications. Two frameworks for the development of software for such DSPs (particularly using the Motorola DSP 56000 and 96000 families) have been developed in the Berkeley DSP Design group; they are called Gabriel [3][10] and Ptolemy [1][4]; and these can be extended for the simultaneous design and simulation of hardware and software, as is shown in this paper.

## 2.0 A Case Study

This section describes the telephone channel simulator that has been selected for this case study, and justifies its selection as a suitable case to demonstrate the use of Ptolemy for hardware/software codesign [7].

### 2.1 The Telephone Channel Simulator

A telephone channel simulator models the response of a telephone channel by generating impairments such as linear distortion, frequency offset, phase jitter, non-linear distortion, and noise [8] (as characterized by the EIA-496-A standard [6]). It is used by voiceband data modem designers to test the performance of modems. Satisfactory performance of modems under these impairment conditions provides robust assurance of modem performance on most telephone lines in the public-switched network of the United States. The algorithm for the telephone channel simulator is shown in Figure 1.

The simulator could be designed using Motorola DSP56000s for most of the signal processing. The hardware model for such a system would typically require, besides DSPs, components such as codecs and glue logic.

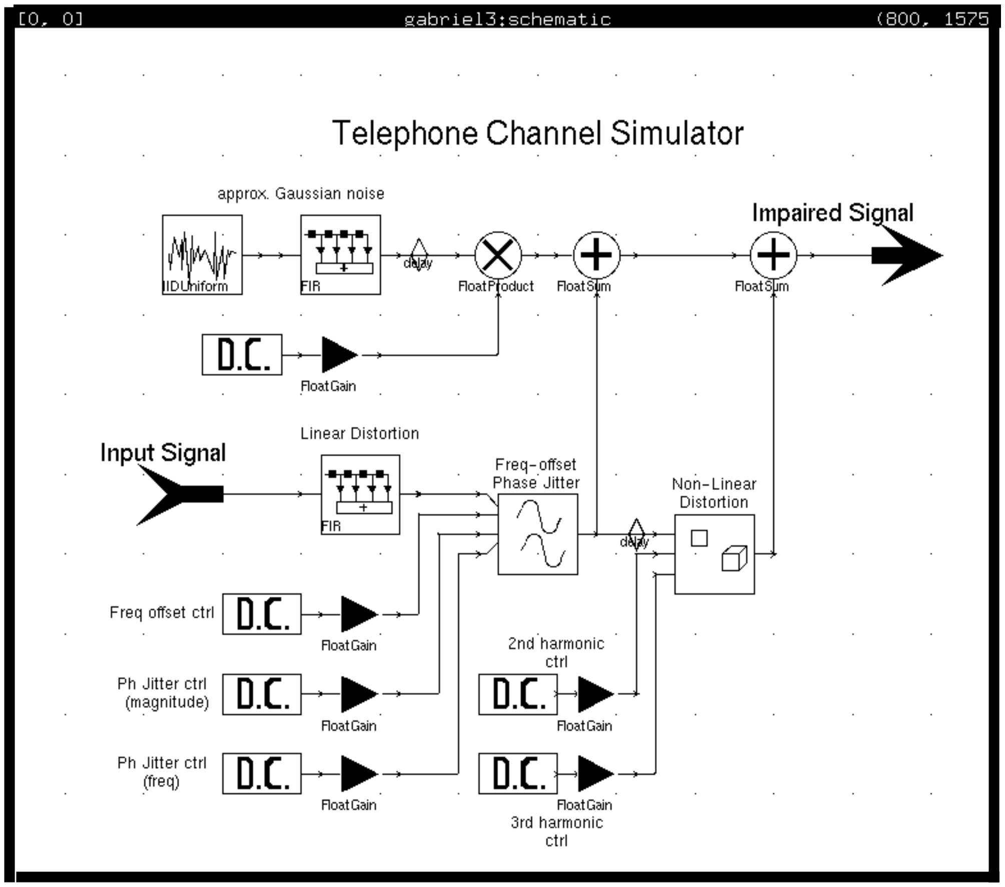


FIGURE 1. Signal Flow Graph for the Telephone Channel Simulator.

## 2.2 Selection of Test Case

The telephone channel simulator, shown in Figure 1 and described earlier in Section 2.1, requires a modest amount of signal processing and represents a ‘real’ system that would benefit from the advantages that codesign offers — low development and production costs and a reduced time to market, while meeting required performance goals. Under technology available today, it may need multiple processors for the signal processing, hence it also brings up many of the issues involved with the design of hardware and software for multiprocessor DSP systems. It is hence critical to design the hardware, generate the software from the algorithm, and simulate the hardware system that executes this software — all within a unified environment. Commercial systems can independently either generate code for DSPs or permit the design and simulation of hardware — they do not provide much interaction between these two capabilities. Ideally, the software synthesis should be retargettable; it should support iteratively changing the hardware configuration and the algorithm. Ptolemy can support all of these requirements.

For all these reasons, the telephone channel simulator has been selected as the test case for the hardware/software codesign study under Ptolemy.

The organization for the rest of the paper is as follows: Section 3.0 briefly describes the Ptolemy simulation environment and its modeling and synthesis capabilities. The codesign methodology is explained (with respect to a case study) in some detail in Section 4.0. Finally, conclusions are drawn in Section 5.0.

## 3.0 The Ptolemy Simulation Environment

Ptolemy is an environment for simulation and prototyping of heterogeneous systems. It uses object-oriented software technology to model each subsystem in a natural and efficient manner, and has mechanisms to integrate these subsystems into a whole.

### 3.1 Internal Structure of Ptolemy

Figure 2 shows the structural components of Ptolemy. The basic unit of modularity in Ptolemy is the Block. Portholes provide the standard interface through which Blocks communicate. A Block contains a module of code (the “go()” method) that is invoked at run-time, typically examining data present at its input Portholes and generating data at its output Portholes. The invocation of “go” methods is directed by a Scheduler that determines the operational semantics of a network of Blocks. Blocks communicate using streams called Particles, which form the base type for all messages passed. The Geodesic class establishes the connection between Portholes. The Plasma class manages the reclamation of the used Particles.

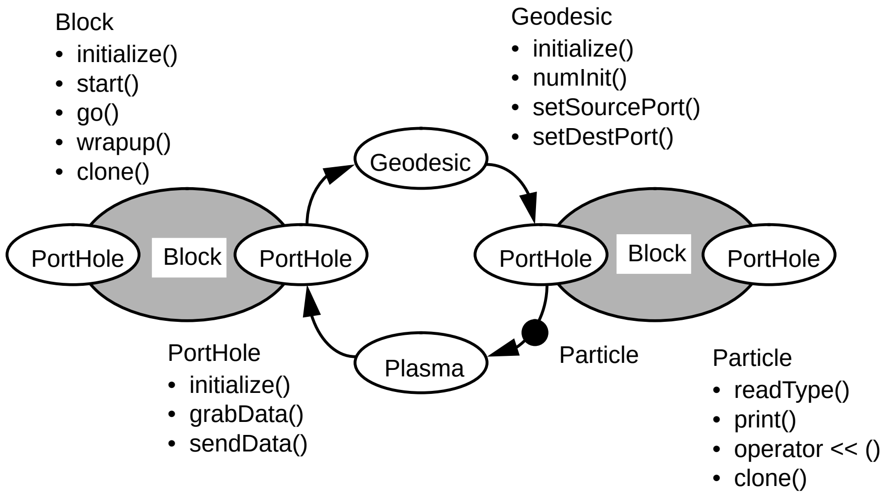


**FIGURE 2. Block objects in Ptolemy send and receive data encapsulated in Particles to the outside world through Portholes. Buffering and transport is handled by the Geodesic and garbage collection by the Plasma.**

The lowest level (atomic) objects in Ptolemy are of the type Star, derived from Block. A Galaxy, also derived from Block, contains other Blocks internally. A Galaxy may contain internally both Galaxies and Stars. A Target, also derived from Block, controls the execution of an application. In a simulation-oriented application, it will typically invoke a Scheduler to manage the order in which Star methods are invoked. For a synthesis-oriented application it can synthesize assembly code for a programmable DSP, invoke the assembler etc. A Universe, which contains a complete Ptolemy application, is a type of Galaxy.

## 3.2 Heterogeneous Simulation using Ptolemy

Ptolemy accomplishes the goal of multiparadigm simulation by supporting a plethora of different design styles called Domains. A Domain realizes a computational model appropriate for a particular type of subsystem. A Domain in Ptolemy consists of a set of Blocks, Targets, and associated Schedulers that conform to a common computational model — the operational semantics govern how Blocks interact with one another. The Domain and the mechanism of co-existence of Domains are the primary features that distinguish Ptolemy from otherwise comparable systems such as Comdisco's SPW and Bones and Mentor Graphic's DSPstation. Some of the domains that are currently supported include 'Synchronous Data Flow' (SDF) [9], 'Dynamic Dataflow' (DDF), 'Discrete Event' (DE), and the 'Digital Hardware Modeling Environment' (Thor [13]).The SDF and Thor domains that are used in the application discussed in this paper are now described in further detail.

**Synchronous Data Flow (SDF) Domain:** SDF [9] is a data-driven, statically scheduled domain. "Data-driven" means that the availability of Particles on the inputs of a star enables it. Stars with no inputs ("sources") are always enabled. "Statically scheduled" means that the firing order of the stars is determined only once during the start-up phase and this schedule is periodic. The SDF domain supports simulation of algorithms, and also allows functional modeling of components such as filters and signal generators.

**Thor Domain**: The Thor Domain implements the Thor simulator, which is a functional simulator for digital hardware developed at Stanford [13]. It supports the simulation of circuits with abstraction levels from the gate level to the behavioral level. Thor hence provides Ptolemy with the ability of simulating digital components ranging in complexity from simple logic gates to programmable DSP chips. The Thor domain could be replaced by a VHDL domain that could support hardware simulation and synthesis.

The Domain class by itself gives Ptolemy the ability of modeling different types of systems in a natural and efficient manner. It also possible to mix these descriptions at the system level to develop a heterogeneous system. Towards this end, Ptolemy allows different Domains to co-exist at different levels of the hierarchy. Within a domain, it is possible to have blocks containing foreign domains. The manner in which these domains co-exist is now described.

Figure 3 shows the top view of a Universe associated with a certain Domain XXX, associated with which are the XXXStars and the XXXScheduler. A foreign subsystem is added to this, which belongs to Domain YYY and has its own set of YYYStars and a YYYScheduler. This foreign subsystem is called a XXXWormhole. A Wormhole is essentially a Block, which appears externally to be a Star (it obeys the operational semantics of the external domain and appears to be atomic to the external domain), but which internally consists of an entire foreign Universe (Scheduler and Stars). A Wormhole can be introduced into the XXX domain without any need for the XXXScheduler knowing of the existence of the YYY domain. The key to this interoperability is the interface between the internal structure of a Wormhole and its external environment. This interface is called the EventHorizon. By providing a “universal” EventHorizon, each domain needs to take care of just the conversion to and from itself to the universal type. More details on the operation of EventHorizons can be found in [1][4].

## 3.3 Hardware Modeling under Ptolemy

Ptolemy supports the modeling and simulation of a variety of hardware components using multiple domains. The Thor domain is used to model digital components — ranging in complexity from simple logic gates to DSP chips [2].

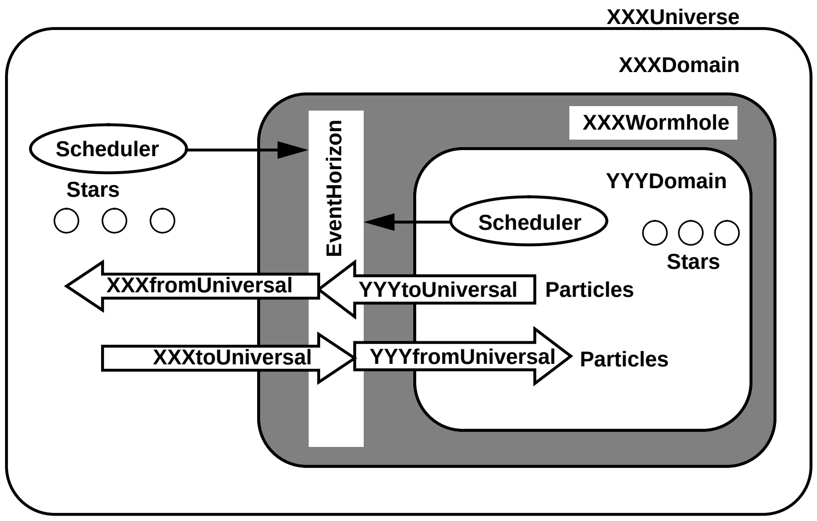


**FIGURE 3. The universal EventHorizon provides an interface between the external and internal domains.**

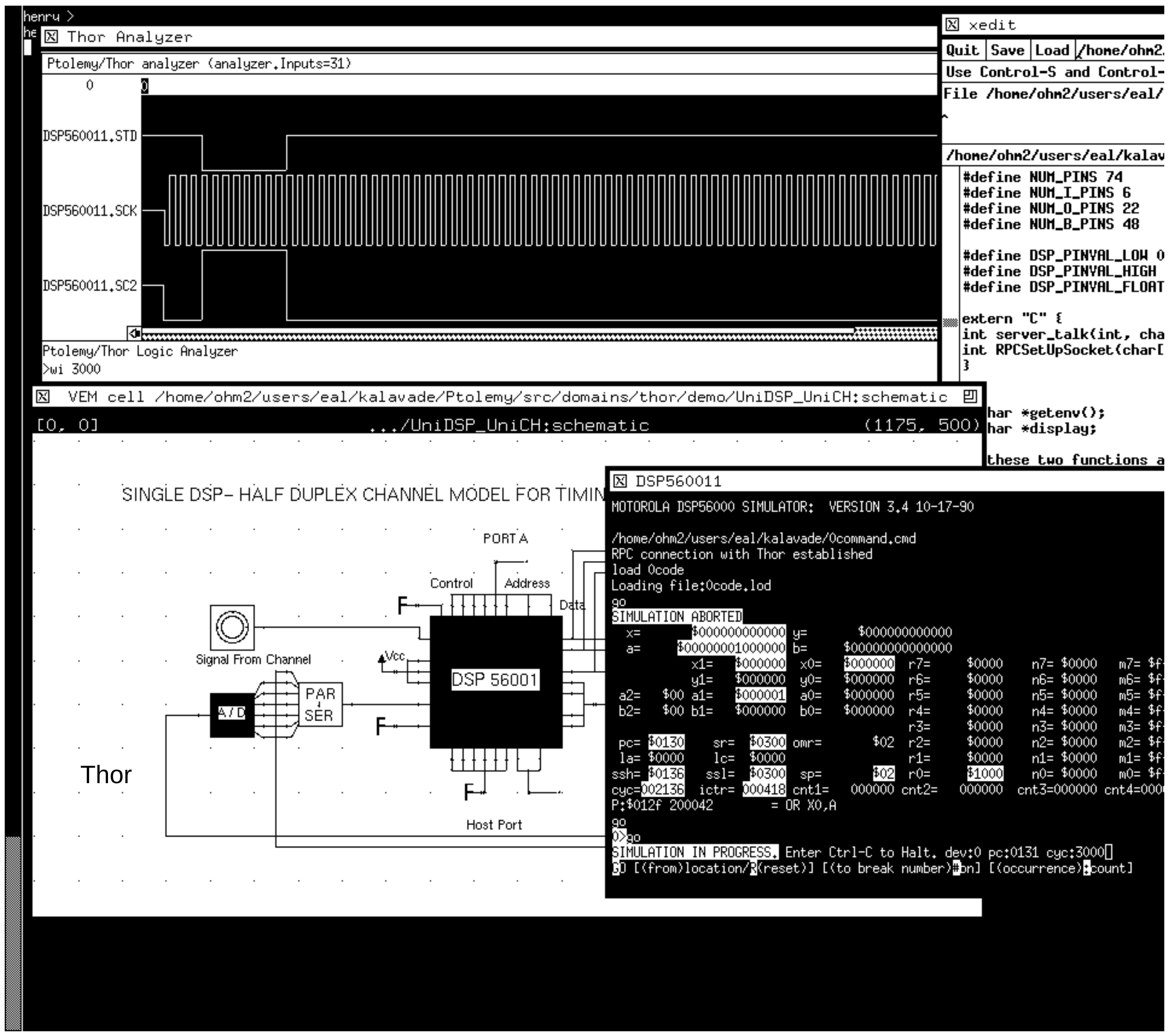
A Thor model for the Motorola DSP56000 has been developed under Ptolemy. The “start()” method of the DSP Star under Thor simply establishes a socket connection with *Sim56000*, Motorola’s stand-alone simulator for the DSP56000 [5]. *Sim56000* is different from most other processor simulators. It is a complete “behavioral simulation” of the processor, not just a “instruction set simulator” or a “bus functional model”. This means that it accurately models the behavior of each of the processor’s signal pins, while executing the code. Instruction set simulators do not support this feature of modeling the pin-level behavior. Bus functional models just emulate a given pattern of the bus activity, they do not execute any code.

During its “go()” method, this Star translates the logic values present at the processor’s pins into values meaningful to the simulator, transfers them to *Sim56000*, and commands the simulator to advance the simulation by one step. It waits for the simulator to transmit the new logic values back to the processor pins and continues with the rest of the simulation. The simulation can be halted at any time by interrupting the simulator window and intermediate register contents can be examined.

Figure 4 illustrates this behavior. The figure represents the simulation of a single DSP system within Ptolemy. The hardware design consisting of a single processor is developed using the Thor and SDF domains of Ptolemy (lower left). The *Sim56000* that is invoked when the system is run can be seen on the lower right. The window in the top left corner shows a Thor logic analyzer monitoring the output on the serial port of the DSP. The Ptolemy code for the DSP block is shown on the top right corner.

Besides processors and digital logic, it is also necessary to model analog components such as A/D and D/A converters, and filters that operate in conjunction with this digital hardware. These analog components can most conveniently be represented by their “functional models” using the SDF domain. It can be observed that abstract functional modeling of components such as filters is sufficient — detailed behavioral modeling is not needed — because, in the final product it is very likely that an off-the-shelf component will be used. So, a filter can be easily modeled by an abstract model in the SDF domain, which merely models the response of the filter in terms of the transfer function and is not concerned with the physical implementation of the filter.

The Wormhole mechanism discussed above in Section 3.2 is used to mix the data-driven, statically-scheduled SDF models of analog components with event-driven, logic-valued Thor models of digital components within a single simulation. This concept is used, for instance, for the modeling of an A/D converter within a Thor domain application containing DSPs and glue logic. Thus, analog and digital hardware modeling at different levels of abstraction is possible using Ptolemy.


henru >
Thor Analyzer
Ptolemy/Thor analyzer (analyzer.Inputs=31)
0
DSP560011.STD
DSP560011.SCK
DSP560011.SC2
Ptolemy/Thor Logic Analyzer
>wi 3000
VEM cell /home/ohm2/users/eal/kalavade/Ptolemy/src/domains/thor/demo/UniDSP_UniCH:schematic
[0, 0] .../UniDSP_UniCH:schematic (1175, 500)
SINGLE DSP- HALF DUPLEX CHANNEL MODEL FOR TIMIN
PORT A
Control
Address
Data
Signal From Channel
Vcc
PAR SER
A/D
DSP 56001
Host Port
Thor
xedit
Quit
Save
Load
DSP560011

```
/home/ohm2.
Use Control-S and Control-
File /home/ohm2/users/eal/
^

/home/ohm2/users/eal/kalav
#define NUM_PINS 74
#define NUM_I_PINS 6
#define NUM_O_PINS 22
#define NUM_B_PINS 48

#define DSP_PINVAL_LOW 0
#define DSP_PINVAL_HIGH
#define DSP_PINVAL_FLOAT

extern "C" {
int server_talk(int, cha
int RPCSetUpSocket(char[
}

har *getenv();
har *display;

these two functions a
```

```
MOTOROLA DSP56000 SIMULATOR:  VERSION 3.4 10-17-90

/home/ohm2/users/eal/kalavade/0command.cmd
RPC connection with Thor established
load 0code
Loading file:0code.lod
go
SIMULATION ABORTED
  x=        $000000000000 y=        $000000000000
  a=      $00000001000000 b=      $00000000000000
          x1=   $000000  x0=   $000000  r7=     $0000    n7= $0000   m7= $f
          y1=   $000000  y0=   $000000  r6=     $0000    n6= $0000   m6= $f
 a2=   $00 a1=   $000001  a0=   $000000  r5=     $0000    n5= $0000   m5= $f
 b2=   $00 b1=   $000000  b0=   $000000  r4=     $0000    n4= $0000   m4= $f
                                         r3=     $0000    n3= $0000   m3= $f
 pc= $0130   sr=  $0300 omr=       $02  r2=     $0000    n2= $0000   m2= $f
 la= $0000   lc=  $0000                  r1=     $0000    n1= $0000   m1= $f
ssh= $0136  ssl=  $0300  sp=       $02  r0=     $1000    n0= $0000   m0= $f
cyc=002136  ictr= 000418 cnt1=   000000 cnt2=   000000  cnt3=000000 cnt4=000
P:$012f 200042       = OR X0,A
go
0>go
SIMULATION IN PROGRESS. Enter Ctrl-C to Halt. dev:0 pc:0131 cyc:3000[]
GO [(from)location/R(reset)] [(to break number)#bn] [(occurrence):count]
```

**FIGURE 4. An illustration of the Ptolemy simulation environment. A hardware design (bottom left) containing a programmable DSP can be developed using the Thor and SDF Domains. When the DSP Star (c++ code can be seen in upper right window) is run, it invokes the Motorola DSP56000 Simulator (bottom right) that executes the code. Timing verification is possible using the Thor Logic Analyzer (top left).**

## 3.4 Software Synthesis under Ptolemy

Ptolemy is capable of synthesizing reasonably optimized assembly code for programmable DSPs using techniques developed in the Gabriel system [3][10]. Descriptions of the hardware (the number of processors, their connectivity, the interprocessor communication strategy, and the IPC delays) and the algorithm block diagram are inputs to the code generator. The code generator contains a retargettable scheduler [12] and a mechanism for DSP assembly code synthesis for multiple processors. The scheduler performs scheduling and routing simultaneously to account for irregular interprocessor interconnections and schedules all computations and communications to eliminate shared resource contention. The code generator thus partitions the program to match the available hardware, and schedules it after considering the physical hardware and the interprocessor-communication and synchronization overhead. This is also retargettable — the hardware description may be changed, and code will be generated automatically for this new configuration, taking into account the changed number of processors or their connectivity.

More highly optimized code generation than that accomplished by Ptolemy has been shown to be viable [11].

## 4.0 Co-Design Methodology

A general approach to codesign is first discussed with reference to Figure 5. Based on this codesign methodology, the design of the telephone channel simulator is next explained in Section 4.1 and Section 4.2.

The first step in the general design approach (step “1” in Figure 5) is to obtain the specifications for the application. This is followed by a high-level algorithmic and functional simulation (step

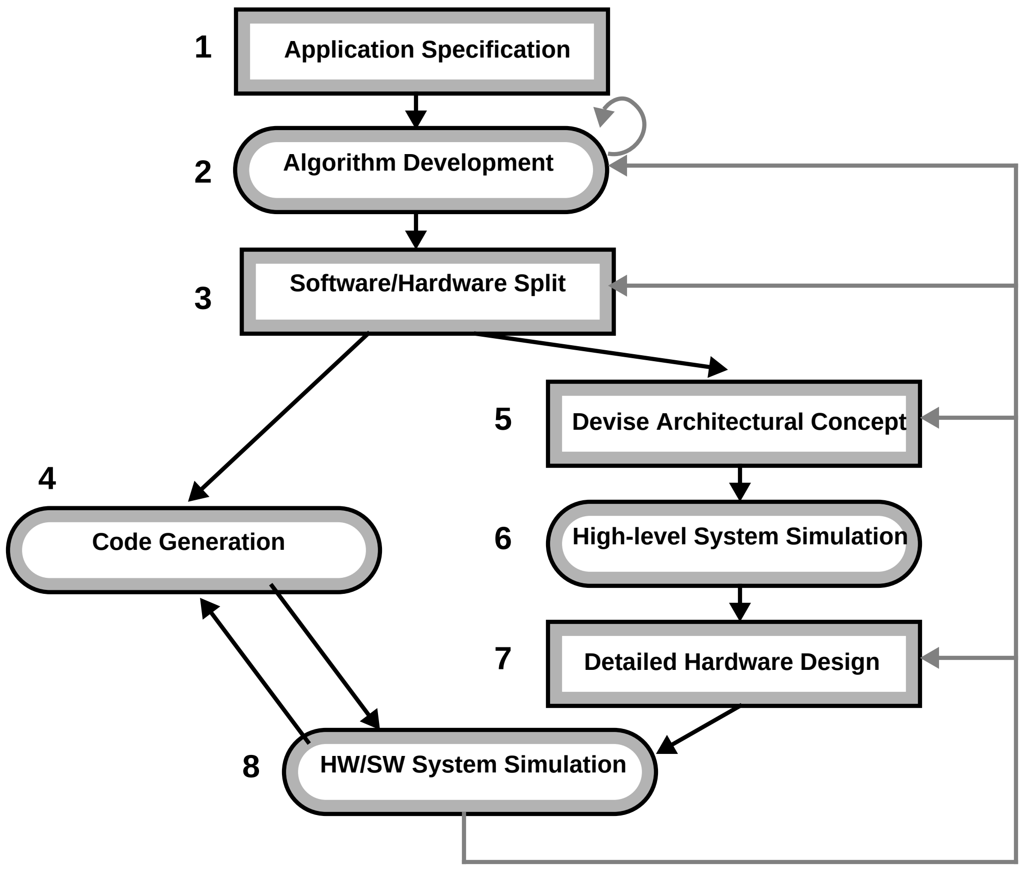


FIGURE 5. Codesign Methodology Using Ptolemy

“2” in Figure 5) using the SDF domain. The functional block diagram (such as that shown in Figure 1) is simulated to verify algorithmic correctness. A hardware/software split (step “3” in Figure 5) is then carried out and parts of the algorithm are assigned to hardware and others to software. A hardware simulation (steps “5-7” in Figure 5) for the configuration is developed, and using this hardware configuration information, software is synthesized (step “4” in Figure 5). The combined hardware/software system is then simulated (step “8” in Figure 5) and verified for functional and timing correctness.

The process may be repeated and various designs for the system can be iteratively developed. The Ptolemy framework allows simultaneous software and hardware development, as well as simulation and evaluation (on the basis of timing constraints and functional correctness) of alternative candidate design configurations.

The following subsections will focus on applications of this codesign approach to the design of a full-duplex telephone channel simulator. In the process, issues and trade-offs associated with the design of multiprocessor systems and system-level partitioning of functionality are studied. The final design choice is made after evaluating the trade-offs associated with each design option.

As an initial logical choice, we design the full-duplex (bi-directional) telephone channel simulator around two DSPs, each one impairing the signal in either direction.

## 4.1 Multiprocessor Options

The specifications of the algorithm (step 1 and 2 of Figure 5) are first obtained. As a first cut, the algorithm is so partitioned (step “3” of Figure 5) that the DSPs execute the entire algorithm in software. The hardware thus consists of two DSPs as well as codecs and other glue logic, while the software is the assembly code (corresponding to the block diagram shown in Figure 1) running on the two programmable DSP 56000s. As a part of the design of the architecture (step “5” in Figure 5), interprocessor communication issues are explored.

- **Design 1 (Shared Memory)**: This approach uses two DSPs for the full duplex channel simulator, where the DSPs are configured to communicate through a shared memory. Figure 6 shows the Ptolemy configuration for this system. 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 to the SDF model of computation. The bottom windows show the Thor and SDF domain hardware design (steps 5-7 of Figure 5) containing two programmable DSPs communicating through a dual-ported

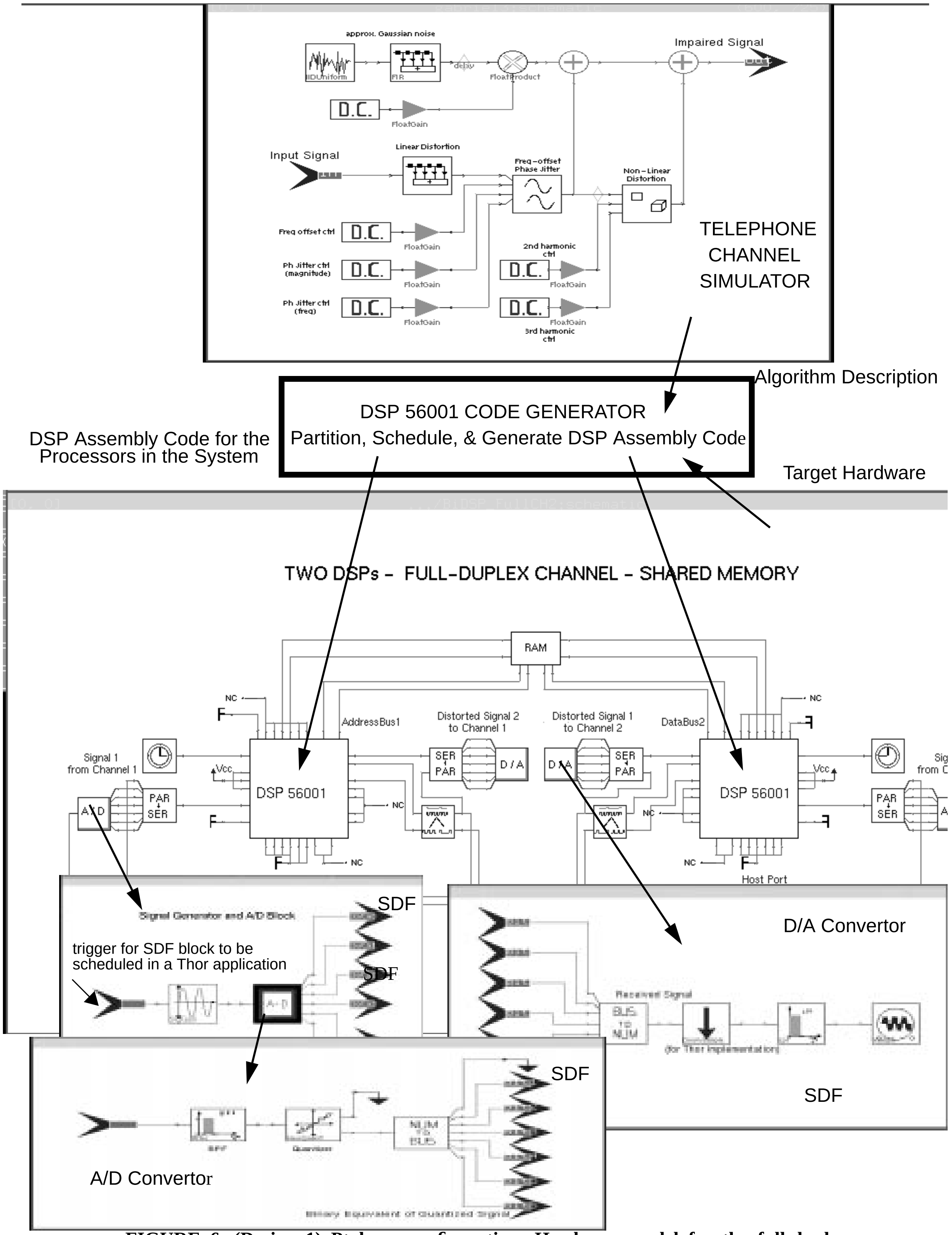


**FIGURE 6. (Design 1) Ptolemy configuration: Hardware model for the full-duplex channel simulator using two DSPs and a shared memory, algorithm description, and the Code Generator.**

shared memory. The full-duplex model consists of the signal from one end of the channel being filtered, quantized, and sent to the first DSP that impairs it. The second DSP reads this data from the shared memory and sends it to the other end of the channel. Similar processing is done in the reverse direction.

The DSP, clock, and other glue logic are implemented in Thor, the parent domain. The A/D and D/A comprising the codec are modeled functionally as SDF blocks nested as Thor Wormholes. The A/D is abstracted as a transmit filter followed by a quantizer. The Stars NumToBus and BusToNum, as shown in Figure 6, provide a conversion between the integers sent/received by the SDF domain and the logic-valued bits understood by the Thor domain.

The algorithm description, along with the hardware description is used by the code generator (step 4 of Figure 5) to partition and schedule the blocks and generate assembly code for the processors in the system. This hardware system is then simulated (step 8 of Figure 5) with the two processors running the code generated by the code generator.

The development of such mixed-domain applications calls for consideration to data-type compatibility and scheduling issues between domains. For instance, the SDF domain deals with integer and floating point Particles while the Thor domain can interpret only 0,1,Z(float), and U(undefined) data types. So, any data passed between SDF and Thor needs to be converted to a binary representation such as that done by the SDFNumToBus block. Also, SDF is a data driven scheduler, which means that an SDF block gets triggered for execution only when the right number of particles are available on all of its inputs. An SDF “source” star however, is always ready for execution when in an SDF universe. Thor, on the other hand, is event driven, which means that the arrival of any particle on any input triggers a new event. When an SDF source block is nested as a ThorWormhole, hence, it needs to be externally triggered (as can be seen by the input provided to the signal generator block in Figure 6). This input data is used solely for the purpose of triggering the wormhole for execution — the data itself is discarded.

- **Design 2 (Serial Port):** An alternative design shown in Figure 7 is developed by iterating through steps (4-8) in Figure 5 where the DSPs now communicate using their serial port instead of the shared memory. As seen in the figure, the signal from one end of the channel is received by the first DSP (DSP1), which impairs it and sends it over the serial port to the second DSP (DSP2), which sends it out to the other end of the channel. Meanwhile, a similar impairment is carried out on the reverse direction for the signal transmitted by DSP2 and received by DSP1. The other blocks such as the A/D and D/A are identical to those developed in Design 1.

  The changed hardware configuration (i.e. the modified IPC delay) is provided to the code generator and the code generator partitions and schedules the code again (“4” in Figure 5).

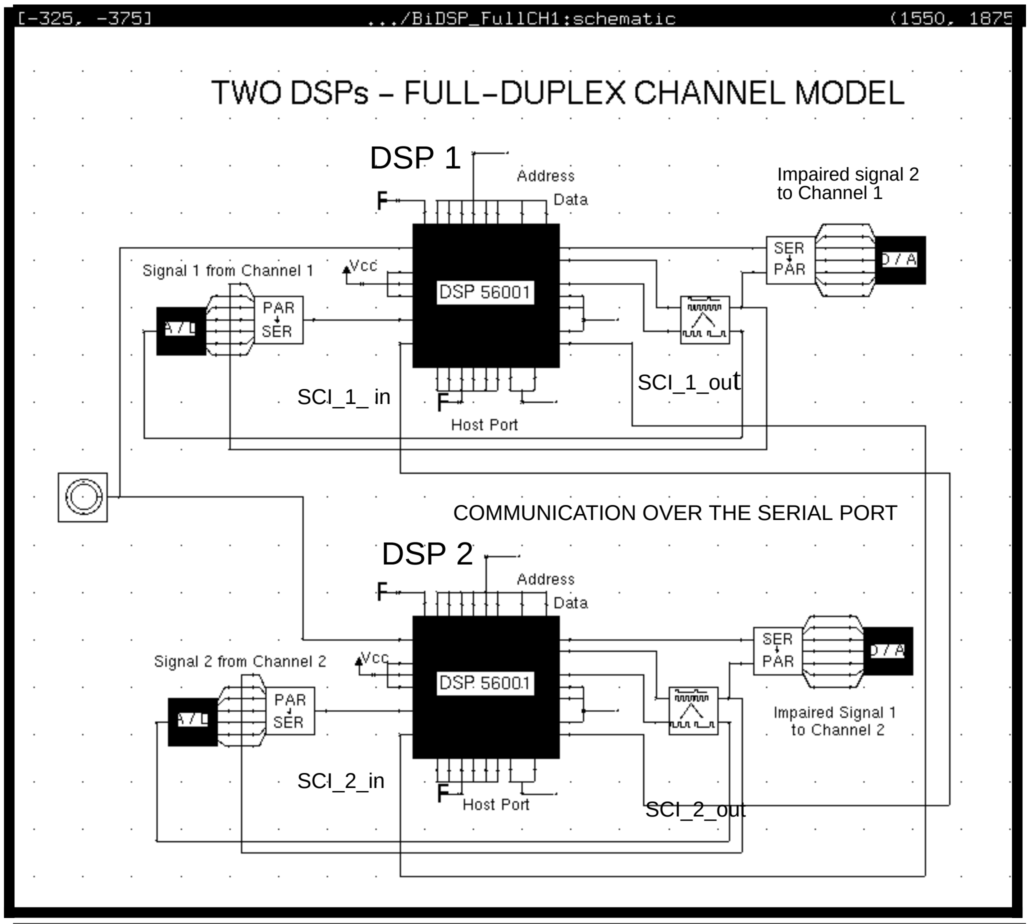


**FIGURE 7. (Design 2) The hardware design for the full-duplex channel simulator using two DSPs and the serial port for communication.**

These designs illustrate the ease with which various interprocessor communication strategies (and the subsequent code partitioning) can be handled within Ptolemy. These two multiprocessor systems (Design 1 and Design 2) are evaluated with respect to performance, IPC overhead, and system cost. These evaluations are discussed further in Section 4.3.

## 4.2 System-level Partitioning Options

In order to reduce the cost of the design, an alternative hardware design possibility, using a single DSP, is now explored. Preliminary timing analysis shows that one DSP cannot handle the computational load of the bi-directional impairments for the full-duplex channel simulator. So, retracing to the hardware/software split (step “3” in Figure 5), the algorithm partitioning is redone. The computationally intensive blocks such as linear distortion and noise generation are possible candidates for migration from DSP software to custom hardware. Designs based on such alternative partitioning are then developed (5-7) within the same framework.

- **Design 3 (Hardware Noise Generation):** As one alternative, the Gaussian noise generator is shifted to hardware and is implemented by using special-purpose diodes. Generating Gaussian noise is expensive in software, but very inexpensive in analog hardware. This functional migration provides sufficient computation cycles for a single DSP to impair the signal in both directions.

  Figure 8 shows such a single processor model. The signals from two ends of the channel are multiplexed and sent to the DSP which runs the impairment code. The impaired signals from the DSP are then routed to the corresponding ends of the channel. The Gaussian noise generator is implemented in hardware using Gaussian noise generating diodes. For the purpose of simulation, however, it is sufficient to model the Gaussian noise generator as an SDF block. The output Gaussian noise from this block superimposes the impaired signal, and this “noisy” signal is then passed to the D/A.

  Thus, shifting the noise generation to hardware makes it possible to use a single processor for the full duplex channel simulator. Note that a functional model of the noise generator is suffi-

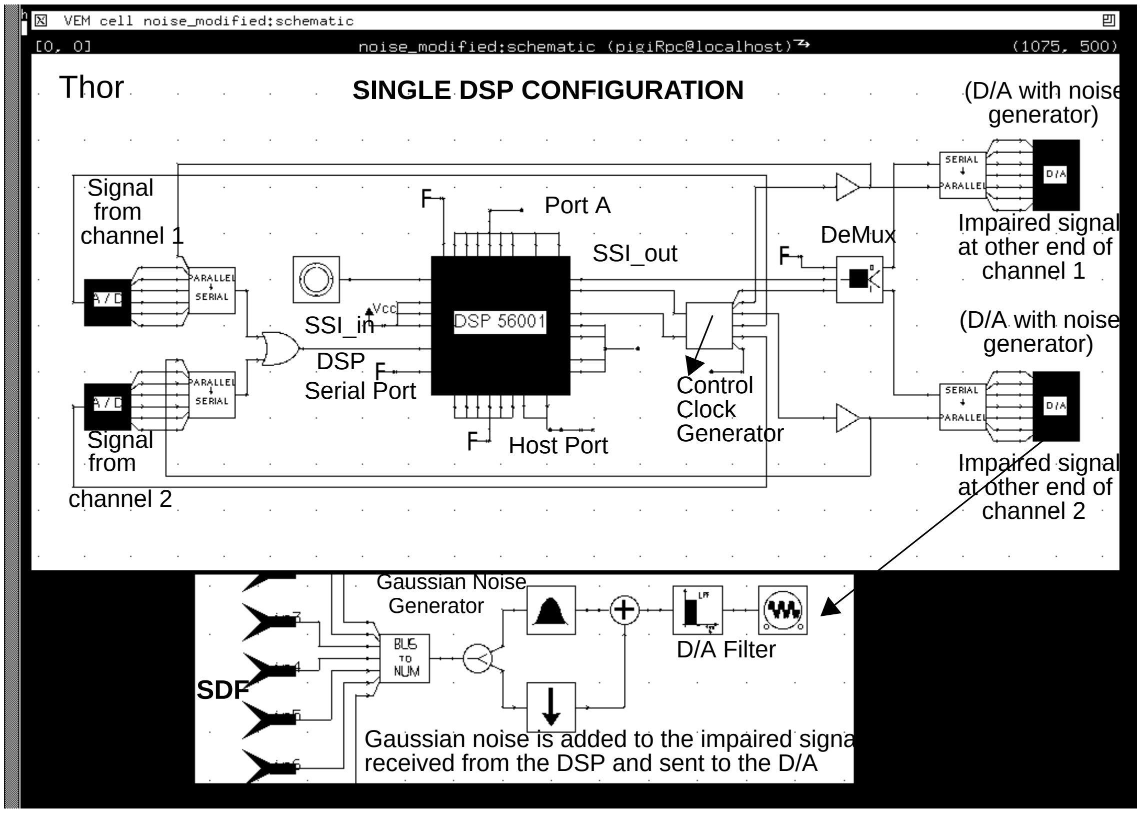

VEM cell noise_modified:schematic
[0, 0]
noise_modified:schematic (pigiRpc@localhost)
(1075, 500)
Thor
SINGLE DSP CONFIGURATION
(D/A with noise generator)
Signal from channel 1
Port A
SSI_out
DeMux
Impaired signal at other end of channel 1
SSI_in
Vcc
DSP 56001
DSP Serial Port
Control Clock Generator
(D/A with noise generator)
Signal from channel 2
Host Port
Impaired signal at other end of channel 2
A/D
PARALLEL SERIAL
SERIAL PARALLEL
D/A
Gaussian Noise Generator
BUS TO NUM
D/A Filter
SDF
Gaussian noise is added to the impaired signal received from the DSP and sent to the D/A

**FIGURE 8. (Design 3) Hardware design for the full duplex channel simulator using a single DSP. The noise generator is implemented in hardware. For the purpose of simulation, the noise generator is implemented as an SDF block generating Gaussian noise. This noise is superimposed on the impaired signal received from the DSP and sent to the D/A.**

cient to demonstrate the functionality and verify the specifications — a "physical" model of the diodes is not necessary. The simulation of this design shows that it meets the timing requirements while still performing the functions of the full-duplex channel simulator.

- **Design 4 (Linear Distortion Variant)**: Shifting much of the linear distortion block to hardware provides another design option (not shown). Much of the linear distortion imposed by a telephone channel is due to switched-capacitor filters in codecs that provide the interface to the digital network. By using similar codecs, we can avoid having to replicate this distortion in software. A much lower order filter is implemented in software to emulate other sources of distortion. The consequent reduction in the filter order reduces the computation load of the DSP. This makes it possible to develop yet another single processor design where the Gaussian noise block is added back to software and Linear Distortion is moved to hardware.

The new algorithm is provided to the code-generator (4) to develop code for these configurations and the resultant systems may be simulated (8) verified for timing and functional correctness. By merely changing the "target architecture" description, code is automatically generated for these different system configurations.

## 4.3 Observations

Earlier sections illustrate that alternative multiprocessor-system designs and system-level partitions can be developed with reasonable ease using Ptolemy. Issues of hardware design, code partitioning, and scheduling can be analyzed. Alternative IPC schemes can be evaluated for cost/performance trade-offs.

The serial communication design (Design 2) has the advantage that it overcomes the need for the expensive dual-ported memory needed by the shared memory design (Design 1); however, it suffers from the drawback that it is not flexible in terms of code-expansion. The relatively slow speed of the serial communication between the DSPs places an upper bound on the performance of the serial communication design. The shared memory design is relatively more flexible and can support higher inter-DSP communication throughput, though at an increased hardware cost. Also, communication through the shared memory requires semaphore synchronization, which imposes an additional burden on the software. Thus, the final design choice may be made on the basis of demands of the user.

System-level design issues regarding the migration of functionality between hardware and software have also been studied. Different components of the algorithm can be implemented either in hardware or software. The trade-offs involved in the use of special-purpose hardware against programmable processors with respect to board space, component cost, and programmability can be compared and contrasted. The single-DSP designs using either the noise-generator diodes (Design 3) or the codec filters (Design 4) are desirable for their low component cost as one processor and its associated hardware is eliminated. The price paid is loss of flexibility in changing these hardware impairments.

The four cases described above indicate the codesign methodology employed under Ptolemy for application-specific system design. Once the design space is explored, the final choice can be made by the user as per the requirements and resources available.

## 5.0 Conclusions

A case study illustrating the use of Ptolemy for codesign has been presented. Ptolemy is seen to be a flexible environment for the development and simulation of functional and behavioral models of hardware and generation of assembly code for the system. DSP-based hardware designs can be developed and provided to the retargettable code generator which produces assembly code corresponding to the block diagram of the algorithm. Alternative multiprocessor hardware configurations using different IPC schemes as well as different system-level designs with alternative hardware/software partitions can be iteratively developed, simulated, verified, and evaluated.

These designs can be compared within a unified design environment. All options are available at a given time to the user. Without the need to build a hardware prototype, the evaluation of the combined software/hardware system is possible. As the software and hardware developments proceed with close interaction, the software is ready at design time. This reduces the time to market.

The key property in Ptolemy that supports this style of design is heterogeneity. Software synthesis, hardware modeling, and algorithm simulation are embodied in a single design environment. Different components of a design can be specified using the design style best suited to them, and yet these components can be mixed.

Work is in progress towards extending this hardware/software design framework for embedded system design, where the hardware consists of DSPs with custom logic, or DSP cores, and the software is the program running on these programmable processors. A netlist description of the hardware configuration can be fed to logic synthesis tools to develop semi-custom ASICs, or used with DSP core designs.

## Acknowledgments

This work has been sponsored by the Semiconductor Research Corporation (contract number 92-DC-008) through its Berkeley Center of Excellence in CAD/IC, and Cygnet, a division of Everex.

## References

[1] "*The Almagest: Manual for Ptolemy Version 0.3.1*", Department of EECS, University of California, Berkeley, CA 94720, USA, January 1992.

[2] J.Bier, "FRIGG: A Simulation Environment for Multiple-Processor DSP System Development", Master's Report, EECS Dept.,Univ. of California, Berkeley 1989.

[3]J.Bier, E.Goei, W. Ho, P.Laplsey, 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.

[4]Joseph Buck, Soonhoi Ha, Edward A. Lee, David G. Messerschmitt, "Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems", *International Journal of Computer Simulation, special issue on "Simulation Software Development".*

[5] *"DSP 56000 Digital Signal Processor User's Manual*", Motorola.

[6]Electronic Industries Association Proposed Standard EIA-496-A Interface Between Data Circuit-Terminating Equipment (DCE) and the Public Switched Telephone Network (PSTN)", Prepared by EIA Technical Subcommittee TR-30.3.

[7]Asawaree Kalavade, "Hardware/Software Codesign using Ptolemy: A Case Study", Master's Report, EECS Dept., Univ. of California, Berkeley, Dec. 1991.

[8]E.A.Lee, D.G.Messerschmitt, "Digital Communications", Kluwer Academic Publishers, pp 128-135.

[9]E.A.Lee, D.G.Messerschmitt, "Synchronous Data Flow", *IEEE Proceedings,* September 1987.

[10]E.A.Lee, W.H.Ho, E.Goei, J.Bier, and S. Bhattacharya, "Gabriel: A Design Environment for DSP", *IEEE Trans. on ASSP*, November, 1989.

[11]Douglas B. Powell, Edward A. Lee, William C. Newman, "Direct Synthesis of Optimized Assembly Code from Signal Flow Block Diagrams", *Proceedings of ICASSP,* San Francisco, March 1992, pp V-553-556.

[12]Gil. C. Sih, "Multiprocessor Scheduling to Account for Interprocessor Communication", Ph.D. Thesis, Electronic Research Laboratory, University of California, Berkeley, CA 94720, April 1991.

[13]"*Thor Tutorial*", VLSI/CAD Group, Stanford University, 1986.