Hardware / Software Codesign Appnote

Abstract

Hardware/Software (HW/SW) Codesign refers to the simultaneous consideration of hardware and software within the design process. Historically, signal processing systems have been designed by specifying the hardware and subsequently making the software fit. On the RASSP program, Lockheed Martin Advanced Technology Laboratories has changed this paradigm by tightly coupling the evaluation of software performance with the selection of hardware architecture and incorporating continuous re-verification throughout the process. Hardware/Software Codesign is the co-development and co-verification of hardware and software through the use of simulation and/or emulation. The emphasis of the RASSP program is on signal processing systems, which provides a very well defined application domain.

Purpose

Although the RASSP process fosters maximum reuse of both hardware and software, new software primitives or custom hardware is often required for an application. When custom hardware is required, HW/SW codesign refers to the process of model generation and the verification of the software on the hardware models prior to hardware build. In the case of new software primitives, HW/SW codesign refers to the process of software generation and verification on a hardware testbed or model(s) of the hardware to ensure both function and timing.

Roadmap

Approved for Public Release; Distribution Unlimited *Dennis Basara*

RASSP Hardware / Software Codesign Application Note

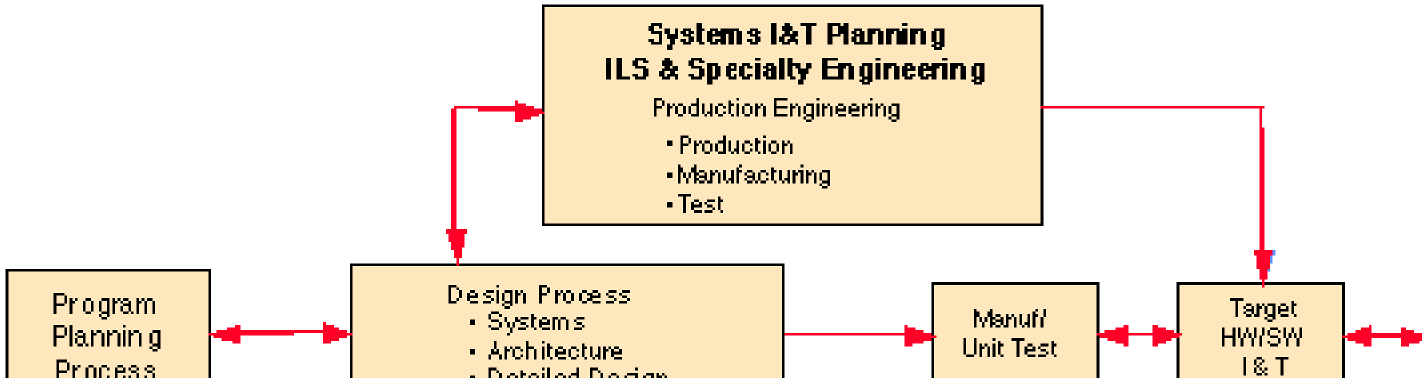
1.0 Executive Summary

1.1 Overview

Hardware/Software (HW/SW) Codesign refers to the simultaneous consideration of hardware and software within the design process. Historically, signal processing systems have been designed by specifying the hardware and subsequently making the software fit. On the RASSP program, Lockheed Martin Advanced Technology Laboratories has changed this paradigm by tightly coupling the evaluation of software performance with the selection of hardware architecture and incorporating continuous re-verification throughout the process. Hardware/Software Codesign is the co-development and co-verification of hardware and software through the use of simulation and/or emulation. The emphasis of the RASSP program is on signal processing systems, which provides a very well defined application domain.

The RASSP process shown in Figure 1 - 1 begins with an architecture-independent data flow graph(s) representing the signal processing algorithms for the intended application. The graphs may initially be non-functional in the sense that they represent the desired data flow but not the functional details. During the process of selecting an architecture, the nodes of the data flow graph(s) are allocated to hardware or software. Allocation to hardware implies that special-purpose hardware will be used, while allocation to software implies that the processing nodes will be implemented on specific programmable processors. This allocation is initially performed using engineering judgement but may be modified as tradeoff studies proceed. The graph nodes allocated to software are mapped to the multiple processors in the architecture, and performance estimates are generated using VHDL token-based simulation. Timing data used in the simulations may either be estimates or benchmarked data on specific processors. During the architecture selection process, alternative hardware architectures are developed and iteratively optimized by modifying either the software mapping or the hardware architecture. Software is automatically generated for all of the mapped partitions and execution time is estimated for the target hardware.

Although the RASSP process fosters maximum reuse of both hardware and software, new software primitives or custom hardware is often required for an application. When custom hardware is required, HW/SW codesign refers to the process of model generation and the verification of the software on the hardware models prior to hardware build. In the case of new software primitives, HW/SW codesign refers to the process of software generation and verification on a hardware testbed or model(s) of the hardware to ensure both function and timing.



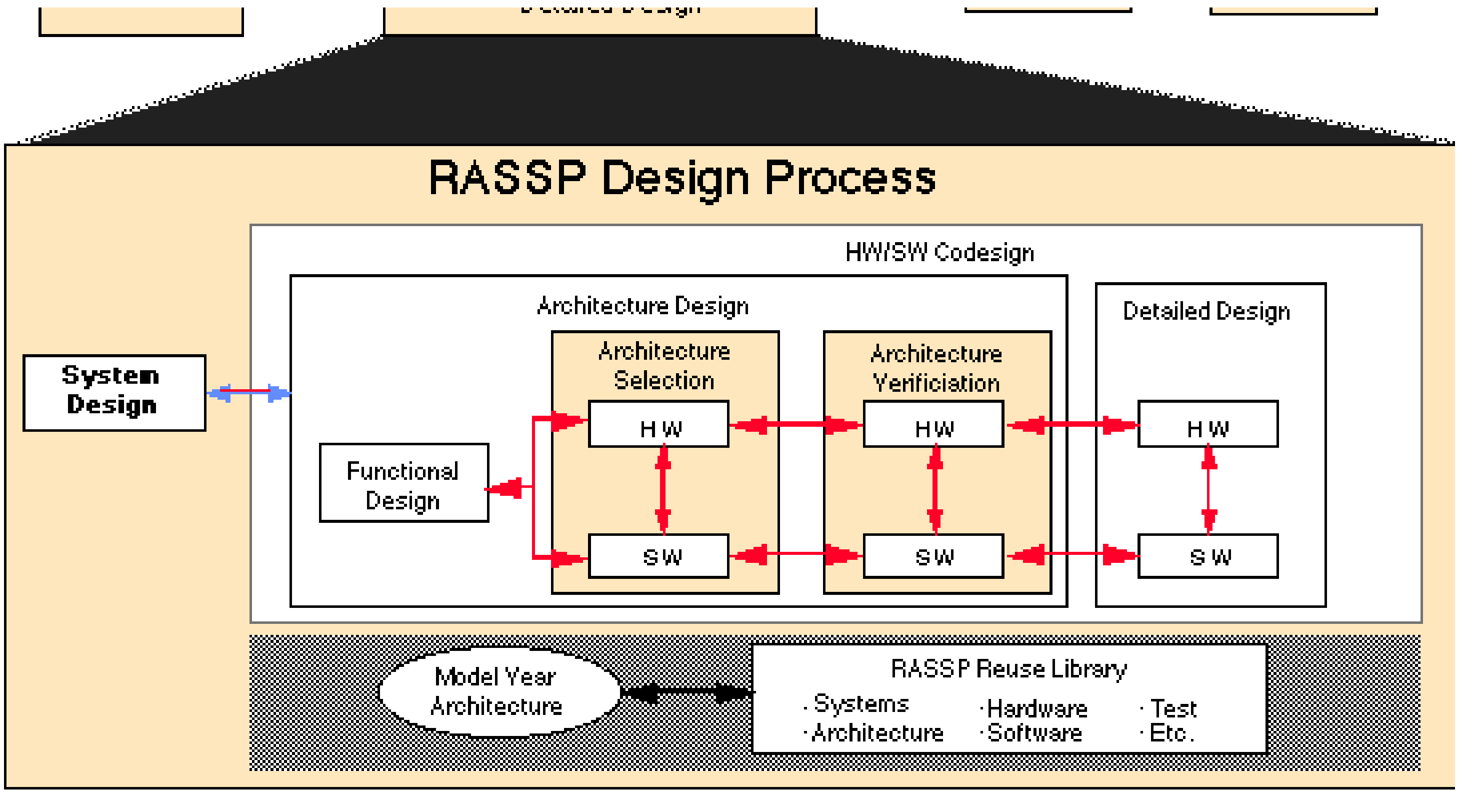


Figure 1 - 1: RASSP Design process.

Lockheed Martin ATL has implemented its Hardware/Software Codesign process as part of an overall RASSP design process. As shown in Figure 1-1, the RASSP design process is composed of three major functional processes called system design, architecture design, and detailed design. Hardware/Software Codesign is implemented from the initial partitioning of functions to hardware and software elements all the way to manufacturing release. At each step in the RASSP process, interactive simulation using hardware and software models is performed at equivalent levels of abstraction to verify behavioral operation.

1.2 The System Design Process

The system design process captures customer requirements and converts these system-level needs into processing requirements (functional and timing). Functional and timing analyses are performed to properly decompose the system-level description. The system process has no notion of either hardware versus software functionality or processor implementation.

The system requirements are translated into simulatable functions, which we refer to on RASSP as an executable specification. This represents the first level at which requirements are specified in a manner that we can readily match to simulators to verify performance and functionality in an automated manner. In this phase, processing time is allocated to functions and functional behavior is defined in the form of algorithms that are executable. At this point, all functions are implementation-independent. The executable specification represents a major portion of the systems design information model.

1.3 The Architecture Design Process

The architecture design process transforms processing requirements into a candidate architecture composed of hardware and software elements. This process initiates the trade-offs between the different architecture alternatives. During this process, the system level processing requirements are allocated to hardware and/or software functions. The architecture process results in a detailed behavioral description of the processor hardware and the definition of the software required for each of the processors in the system. The intent is to verify all of the code during this portion of the design process, ensuring hardware/software interoperability early in the design process.

The architecture process is driven from a data flow graph (DFG) description of the processing which is required. The DFG is an architecture independent description of the processing which must be performed by the signal processor. During the architecture process some of the functionality described in the DFG may be allocated to hardware while other functionality may be allocated to software to be executed on COTS DSPs. The architecture independent version of the DFG represents an application description which is the starting point for Hardware/Software Codesign on any particular architecture of interest.

This application note describes:

- a generic Hardware/Software Codesign process;
- tools and techniques used in the process with emphasis on application development, performance simulation and autocoding;
- the Lockheed Martin ATL approach to Hardware/Software Codesign using the Semi-Automated IMINT Processor (SAIP) RASSP Benchmark as an example; and
- lessons learned.

Next Up Previous Contents
Next: 2 Introduction **Up:** Appnotes Index **Previous:**Appnote Hardware / Software Codesign Index

Approved for Public Release; Distribution Unlimited *Dennis Basara*

RASSP Hardware / Software Codesign Application Note

2.0 Introduction

This remainder of this Application Note is organized as follows. Section 3 describes the RASSP Hardware / Software Codesign Process and its implementation. Included is a limited discussion of how both the application and an evolving architecture influence the way in which HW/SW Codesign is applied. Section 3 also includes a description of the integrated RASSP architecture toolset with a discussion of both the process implemented and the tools used. Section 4 provides an example of the application of the integrated toolset to the development of an application using a COTS architecture. The example discussed is the Semi-Automated IMINT processor (SAIP) development conducted under Benchmark 4. Section 5 provides an early RASSP example of the codesign process applied to a mixed COTS/Custom architecture. The exmple used in this discussion is the Synthetic Aperture Radar application implemented in the RASSP Benchmark 2. Section 6 discusses some of the lessons learned from the RASSP benchmark developments. Section 7 provides additional information and references.

Approved for Public Release; Distribution Unlimited *Dennis Basara*

RASSP Hardware / Software Codesign Application Note

3.0 RASSP Hardware / Software Codesign Process

3.1 Definition and Benefits

HW/SW Codesign refers to the simultaneous consideration of hardware and software in the design of a system, rather than the more common approach of specifying the hardware and constraining the software to fit. The RASSP program therefore defines HW/SW codesign as the co-development and co-verification of hardware and software through the use of simulation and/or emulation. This codesign begins with Functional Design and ends with Detailed Design, as shown in Figure 1- 1, where Detailed Design includes software generation but not hardware fabrication. The principal benefits of HW/SW codesign are that it:

- **Enables mutual influence of both hardware and software early in the design cycle.** Software performance is a key criteria for selecting an architecture.
- **Provides continual verification throughout the design cycle.** As the design progresses through subsequent levels of detail, both hardware and software are continually co-verified to improve design quality. This results in minimizing costly iterations after a design is released to manufacturing.
- **Enables evaluation of larger design trade space.** Interoperability of tools and automation of codesign early in the architecture process significantly improve the ability to consider designs that may otherwise be ignored.
- **Reduces integration and test.** Because hardware and software come from reuse libraries and are co-verified throughout the design process, HW/SW codesign is a significant factor in reducing the resources devoted to integration and test. This is one of the largest contributors to achieving a 4x improvement in the overall process.

3.2 Implementation of Hardware / Software Codesign in RASSP

The RASSP design process is based on true hardware/software codesign and is no longer partitioned by hardware and software disciplines but rather by the levels of abstraction represented in the system, architecture, and detailed design processes. Figure 3 - 1 depicts the RASSP methodology as a library-based process that transitions from architecture independence to architecture dependence after the systems process.

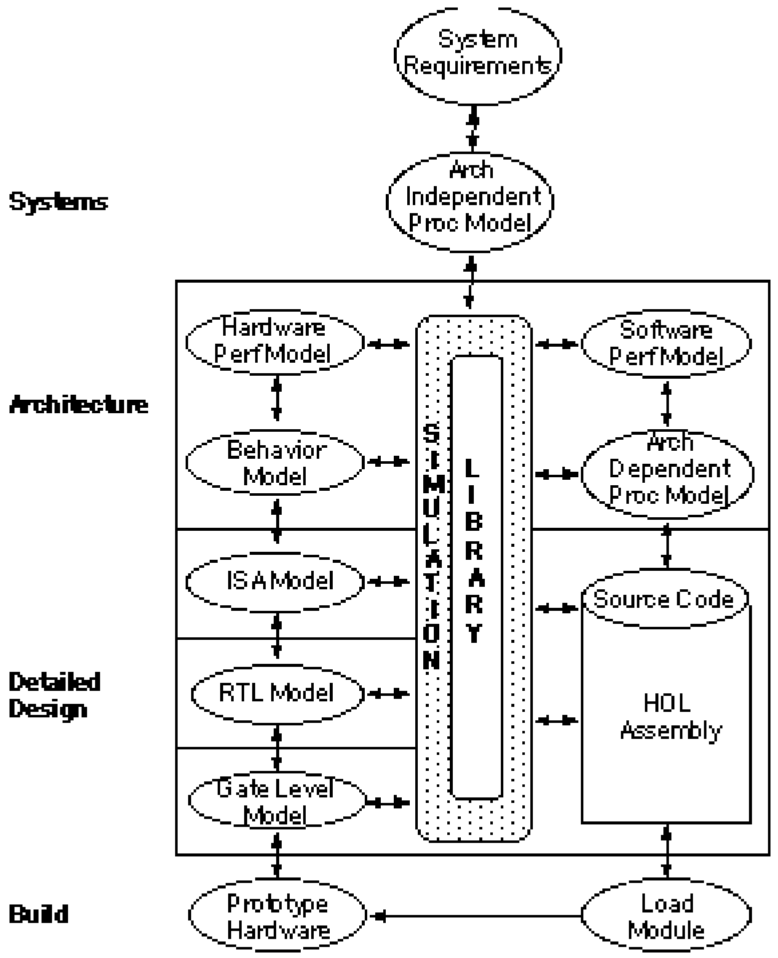


Figure 3 - 1: HW/SW codesign in RASSP design process

In the systems process, processing requirements are modeled in an architecture-independent manner. Processing flows are developed for each operational mode and performance timelines are allocated based on system requirements. Because this level of design abstraction is totally architecture-independent, HW/SW codesign is not an issue.

In the architecture process, the processing flows are translated to data flow graphs and control flow description for subsequent processes. The data flow graph(s) becomes the functional baseline for the required signal processing. The processing described by the nodes in the data flow graph are allocated to either hardware or software as part of the definition of candidate architectures. This becomes the transition to architecture dependence.

The HW/SW allocation is analyzed via modeling of the software performance on a candidate hardware architecture through the use of both software models and VHDL token-based hardware performance models. For selected architectures requiring:

a) new computational elements,
b) interfaces, or
c) communication mechanisms

hierarchical verification is performed using finer grain modeling at the Instruction Set Architecture (ISA) level and below. During the detailed design process, actual software code is verified to the maximum extent possible.

Reuse library support is an important part of the overall process. The methodology supports the generation of both hardware and software models. Software models are validated using appropriate test data, and VHDL hardware models are validated using existing, validated, software models. When a hardware model also requires new software - such as a new interface chip requiring a new driver - the hardware and software models are jointly iterated and modified throughout the design process.

Simulation is an integral part of HW/SW codesign. Figure 3 - 2 shows a top-level view of the simulation philosophy in the RASSP methodology. During the systems process, functional simulation is performed to establish a functional baseline for the signal processing application. This functional baseline is architecture-independent and may be generated using a variety of tools for algorithm development and simulation including MATLAB, PGM Tools or GEDAE™. System designers may make an initial estimate of hardware / software allocation and obtain early cost estimates from tools like PRICE based on these allocations.

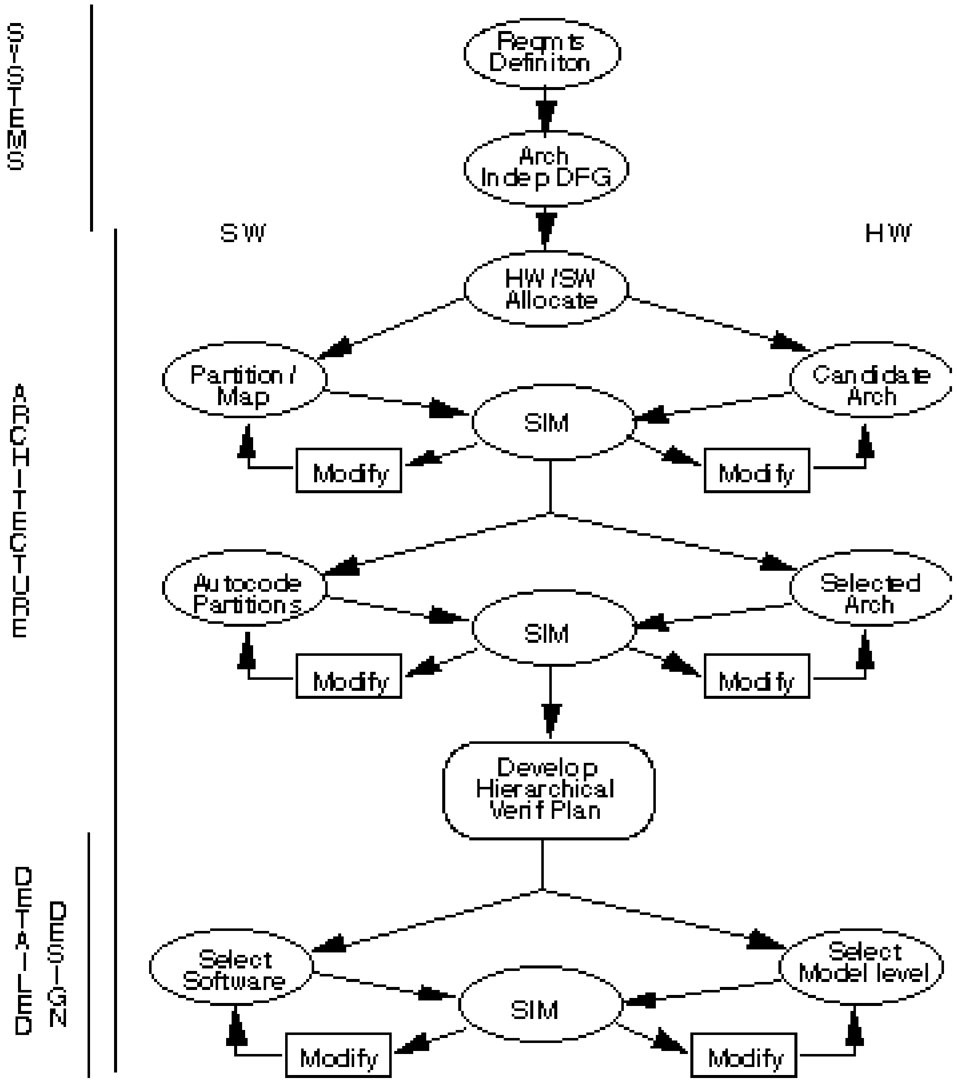


Figure 3 - 2: RASSP simulation philosophy by design process

During the architecture design process, various simulations are performed at differing levels of detail as the design progresses. Early in the process, performance simulations are executed using high-level models of both hardware and software from the reuse library. Software is modeled as execution time equations for software primitives executing on various processors in the architecture. The architecture is described using token-based performance models for both the processing elements and communication elements. This level of simulation facilitates the rapid analysis of a broad range of architectural candidates composed of various combinations of COTS processors, custom processors, and special-purpose ASICs. In addition, many approaches to partitioning and mapping the software for execution on the architecture can be evaluated.

As the architecture process progresses, each of the software graph partitions is automatically translated into a software module for execution on a specific processor in the architecture. Functional simulation is used to verify that the generated code is consistent with the functional baseline. Performance simulation provides the next level of assurance that all throughput requirements are met by using lower level models, including the operating system, scheduling, and support software characteristics. Finally, hierarchical architecture verification of the architecture is established using selective performance and functional simulation at the ISA and/or Register Transfer Level (RTL) level. The goal is to ensure that all architectural interfaces are verified.

In the detailed design process, selective performance and behavioral simulation are performed again. At this point, however, the design has progressed to the point where simulation at the RTL and logic levels is most appropriate. Verification of the designs at this level is necessary prior to release to manufacturing. It is important to note that pieces of the design may be in different stages of the overall process based on the risk analysis performed in each development cycle. For example, if it is obvious to the designers during systems definition that they will need a new custom hardware processor to meet the requirements, they may accelerate the design of the custom processor while the overall signal processor design is still in the architecture process.

3.3 HW / SW Codesign Versus Application and Architecture

Both the application and the evolving architecture for the signal processor influence the way in which HW/SW codesign is applied within the methodology. Table 3 - 1 contains examples of various mixes between COTS and custom solutions for the signal processor. Although the table addresses only hardware, the software may, in a sense, also be custom or COTS. Depending upon the nature of the evolving solution, one or more portions of the design may be defined or implemented concurrently.

3.3.1 All COTS Solution

An all-COTS solution means that in addition to the architecture being all COTS, as shown in line 1, Table 3 - 1, the signal processing data flow graphs can be constructed from existing primitives in the reuse library. There will likely be control software that must be developed, but all of the signal processing can be developed by graphically constructing the data flow graphs . from existing library elements. These data flow graphs will be translated via autocode generation for execution on the target processors under control of the run-time system, which is built with an open Application Programming Interface (API) to standard operating system micro-kernels. When executed on the target hardware, optimized versions of the primitives are utilized. In this type of solution, the HW/SW codesign process is completed when a satisfactory result is obtained from the virtual prototype, which includes both function and timing. This virtual prototype is a VHDL simulation that models the architecture, the run-time system, the operating system characteristics and the autocoded software. It also uses detailed timing data for software performance estimates for the target processors generated during the autocode process.

PE	Comm Element	PE Interface	Comm Network	Board	Comments
1. COTS	CUST	CUST	CUST	CUST	Architecture built from available board Primitives and OS services are available from libraries
2. COTS	CUST	CUST	CUST	CUST	Applicaiton requires custom topoliogy and new board design
3. COTS	CUST	CUST	CUST	CUST	PE needs to be interfaced to new interconnect fabric, new board design requirements
4. COTS	CUST	CUST	CUST	CUST	New communication elements
5. CUST	CUST	CUST	CUST	CUST	Application requires new application specific PE
6. CUST	CUST	CUST	CUST	CUST	Application requires full custom solution

Table 3 - 1: Examples of various COTS versus custom hardware mixes

3.3.2 New primitives Required

Primitives are the bulding blocks from which the data flow graphs are created. A primitive is the smallest element of software which can be allocated to a processor. Although the intent is to provide a wide range of primitives suitable for many applications, there will always be the need for the user to develop and add new primitives to the reuse library. New primitives may be required either because new processing algorithms have been developed or because specialized optimization is required to meet throughput requirements. When new primitives are required, the HW/SW codesign represented by the detailed design portion of Figure 4 may be initiated as soon as the need for the primitives is recognized. Thus the detailed design of a new primitive is performed concurrently with the continuation of the architecture definition process. Initially the architecture definition process may proceed using estimated execution times for the primitives to be developed. The HW/SW codesign of the primitive proceeds by using either existing hardware, if available, or an appropriate level model - such as the ISA model - to both verify functionality and optimize execution times. As the maturity of the primitive implementation progresses, the execution times can be back-annotated to update the performance simulations used in the architecture development process.

3.3.3 Custom Hardware Required

As indicated in Table 3 - 1, there are a variety of reasons why designers may need custom hardware to support the overall architecture being developed. Perhaps they have not interfaced the selected digital signal processor to the desired communication network and require a new processor interface. This may also require that the underlying operating system services, such as communication drivers, be updated to support the new interface. Alternatively, designers may determine that a custom processor is required for a specific part of the application in order to meet the throughput requirements. The designers may initiate the HW/SW codesign represented by the Detailed Design portion of Figure 3 - 2 as soon as they recognize the need for custom hardware. Thus, they may concurrently perform the Detailed Design of the custom hardware and the interface software along with the continuation of the architecture definition process. In these cases, they develop models of the hardware at various levels of abstraction to support the process. They concurrently develop the software required to use the new interface hardware or custom processor and use it in conjunction with the hardware models to verify operation and performance. As designers develop detailed timing data, they use back annotation to update prior performance estimates.

3.4 Integrated RASSP Architecture Toolset

3.4.1 Implemented Process

An implementation of the RASSP architecture design process specifically for COTS architectures is shown in Figure 3 - 3. The signal processing and control requirements shown in the figure are outputs of the systems design process. In the figure, the architecture process is broken into five subprocesses:

- architecture tradeoffs & mapping optimization,
- algorithm definition & optimization,
- testbed execution & optimization,
- command program definition & simulation, and
- target software generation and test.

This overall process permits many of the above activities to be conducted concurrently. The integrated RASSP architecture toolset is based upon three tools, GEDAE™ from Lockheed Martin Advanced Technology Laboratories, Omniview Cosmos™ from Omniview Inc, and ObjectGEODE from Verilog SA.

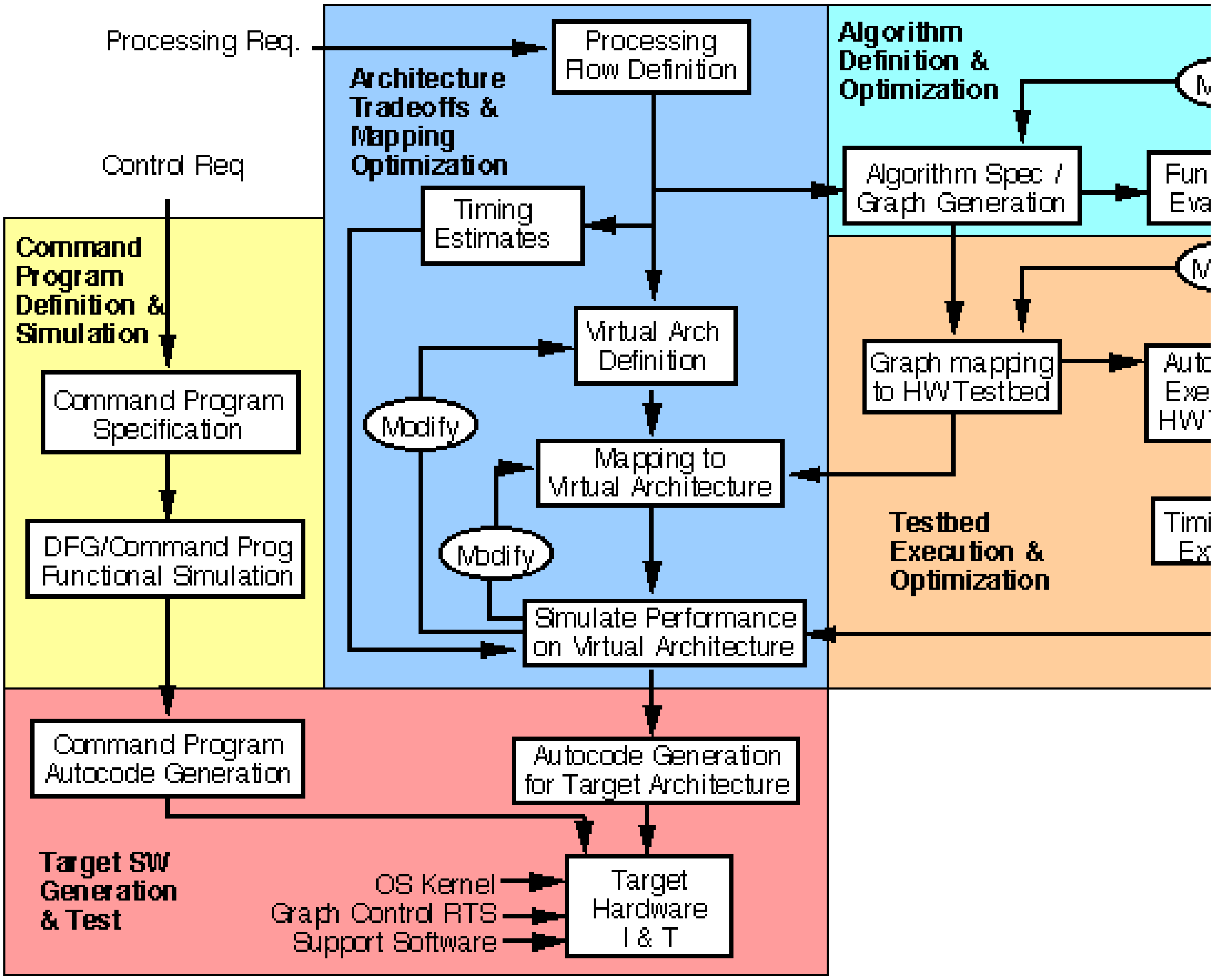

Processing Req.
Control Req
Architecture Tradeoffs & Mapping Optimization
Processing Flow Definition
Algorithm Definition & Optimization
Algorithm Spec / Graph Generation
Timing Estimates
Command Program Definition & Simulation
Virtual Arch Definition
Modify
Mapping to Virtual Architecture
Modify
Simulate Performance on Virtual Architecture
Command Program Specification
DFG/Command Prog Functional Simulation
Graph mapping to HW Testbed
Testbed Execution & Optimization
Command Program Autocode Generation
Autocode Generation for Target Architecture
Target SW Generation & Test
OS Kernel
Graph Control RTS
Support Software
Target Hardware I & T

Figure 3 - 3: RASSP simulation philosophy by design process

3.4.1.1 Architecture Tradeoffs & Mapping Optimization

The first step in the architecture tradeoffs is the generation of the required top level processing flows. It is not necessary to have a complete definition of the specific algorithms to be used, but it is necessary to have a definition and understanding of the data flow required. For early architecture tradeoff analysis, the algorithmic processing can be represented by time estimates (delays) for the computation. It is important however to have as accurate as possible representation of the data flow. The processing flow defines the way in which data must be moved from one algorithmic function to another. Depending on the mapping of the algorithm to the processors in the architecture, the passing of data from one processing function to another may or may not require moving data between physical processors or even between boards in the architecture. It is the data flow and the mapping of the required processing to the individual processors that will define the overall throughput achievable on a particular architecture.

In order to perform architecture tradeoffs, a candidate architecture must be postulated. This can be accomplished graphically as will be shown in subsequent paragraphs. Underlying each of the elements in the hardware architecture is a token-based performance model. As part of the effort required to perform the architecture tradeoffs, any models or variations of existing models which do not exist but are needed to represent candidate architectures must be created. Refer to the Token-Based Performance Modeling application note for the development of these models.

The required processing time for each of the functions in the processing flow must be obtained. Initialy, these times may be estimated and can be updated as implementation details become better defined. After having both a processing flow and a candidate archtecture, various mappings of the required processing to the candidate architecture can be postulated. In each case, the expected performance is simulated and either the architecture or the mapping may be modified to optimize performance. Simulations may be performed using state of the art tools for constructing the simulations and standard VHDL simulators.

3.4.1.2 Algorithm Definition & Optimization

Concurrent with the architecture tradeoffs, developers can be fleshing out the details of the algorithms. Each of the functions in the top level processing flows is replaced with the algorithmic details required by the application. Modern tools provide the ability to graphically construct the algorithm details, simulate the behavior of the algorithms to establish corrrect functionality and optimize the computation. As depicted in Figure 3 - 3, the algorithm definition and optimization is an iterative process which is simplified by graphical programming tools which provide detailed visualizaton capability.

3.4.1.3 Testbed Execution & Optimization

Once the algorithms have been defined, execution of the algorithms (or pieces of the algorithm) on a hardware testbed provides timing data which are more accurate than the original estimates used in the architecture tradeoffs. This new data is used to continually update the timing estimates and revalidate the performance of the selected architecture. The integrated RASSP archtecture toolset simplifies this process. This step can be especially usefull when the developers have access to a single board DSP testbed but the application requires perhaps tens of the boards. The testbed execution provides the accurate algorithm timing estimates and the performance simulation provides the confidence that the overall required data flow does not create crippling communication bottlenecks.

3.4.1.4 Command Program Definition & Simulation

Although the signal processing is the computational heart of an application, the overall control of the signal processing can often be very complex. The integrated RASSP architecture toolset provides assistance in this area as well. Depending on the complexity of the control that is required, it can be beneficial to consider the utilization of graphical programming tools and autocoding for the control as well as the signal processing. The command program definition and simulation indicated in Figure 5 may be accomplished graphically with modern tools that are specifically aimed at graphically specifying, simulating and autocoding software which is best described as a finite state machine. Joint simulation of both the control and signal processing ensures that proper functionality is maintained throughout control state changes. For details on the generation of control software see theAutocoding for DSP Control application note.

A typical large scale conceptual architecture is shown in Figure 3 - 4. It consists of a Signal Processing System (SPS) communicating with a Command and Control System (CCS). The Signal Processing System is that portion of the overall application that performs the high bandwidth "number crunching", is naturally represented by a data-flow model of computation, and typically executes on Digital Signal Processors (DSPs). Intimately related to the SPS, but frequently conceptualized as part of the Command and Control System is the Command Program (CP).

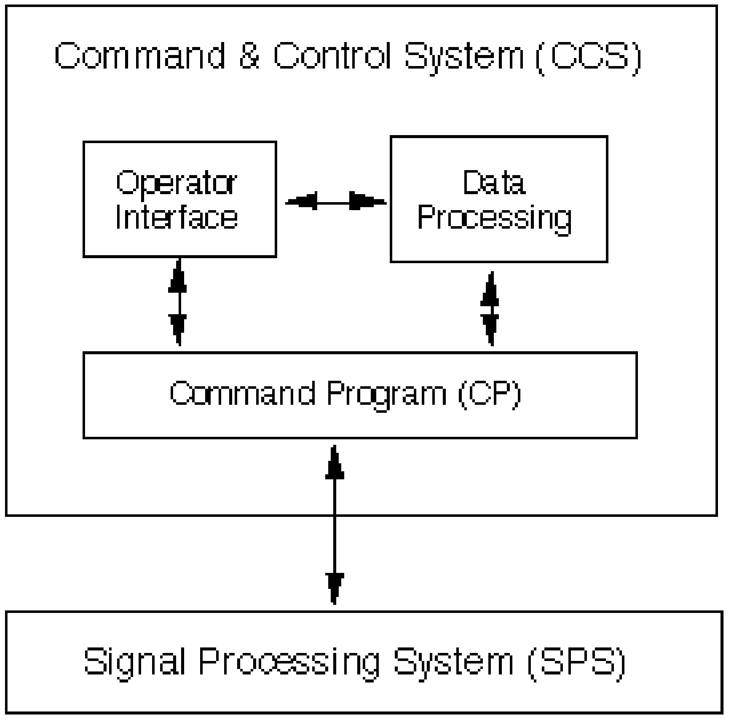

Command & Control System (CCS)
Operator Interface
Data Processing
Command Program (CP)
Signal Processing System (SPS)

Figure 3 - 4: Typical application architecture

The Command Program serves as an interface between the SPS and the rest of the CCS by translating system derived or user inputs into sets of commands that are understood by the SPS and by forwarding SPS results. It is significantly more than a simple data reformating program. The CCS system views the SPS as a collection of domain specific abstractions that are frequently refered to as modes and submodes. For example, in an airborne radar, the CCS system may view the SPS as performing "the weather mode" or "the track submode of the Airborne Target Search and Track Mode". However, the notions of mode or submode are foreign to the SPS. The fundamental concept in the SPS system is that of a graph, while a mode may correspond to a collection of concurrently executing graphs. The SPS is modeled using the Data-Flow paradigm, while the Command Program is often represented by a Finite State Machine. Thus the command program must transform the CCS notions of mode and submode into the SPS notions of graphs and dataflow.

Another representation of the command program and signal processing software development process is shown in Figure 3 - 5. The first step is to capture the set of graphs to be controlled by the command program. This depends only on knowing the set of modes and submodes, which may already be defined in the procurement specification, and the top level assignment of signal processing algorithms to a set of signal processing graphs. Command Program development is not dependent on the detailed implementation of the signal processing graphs whose implementation is shown on the right hand side of Figure 3 - 5.

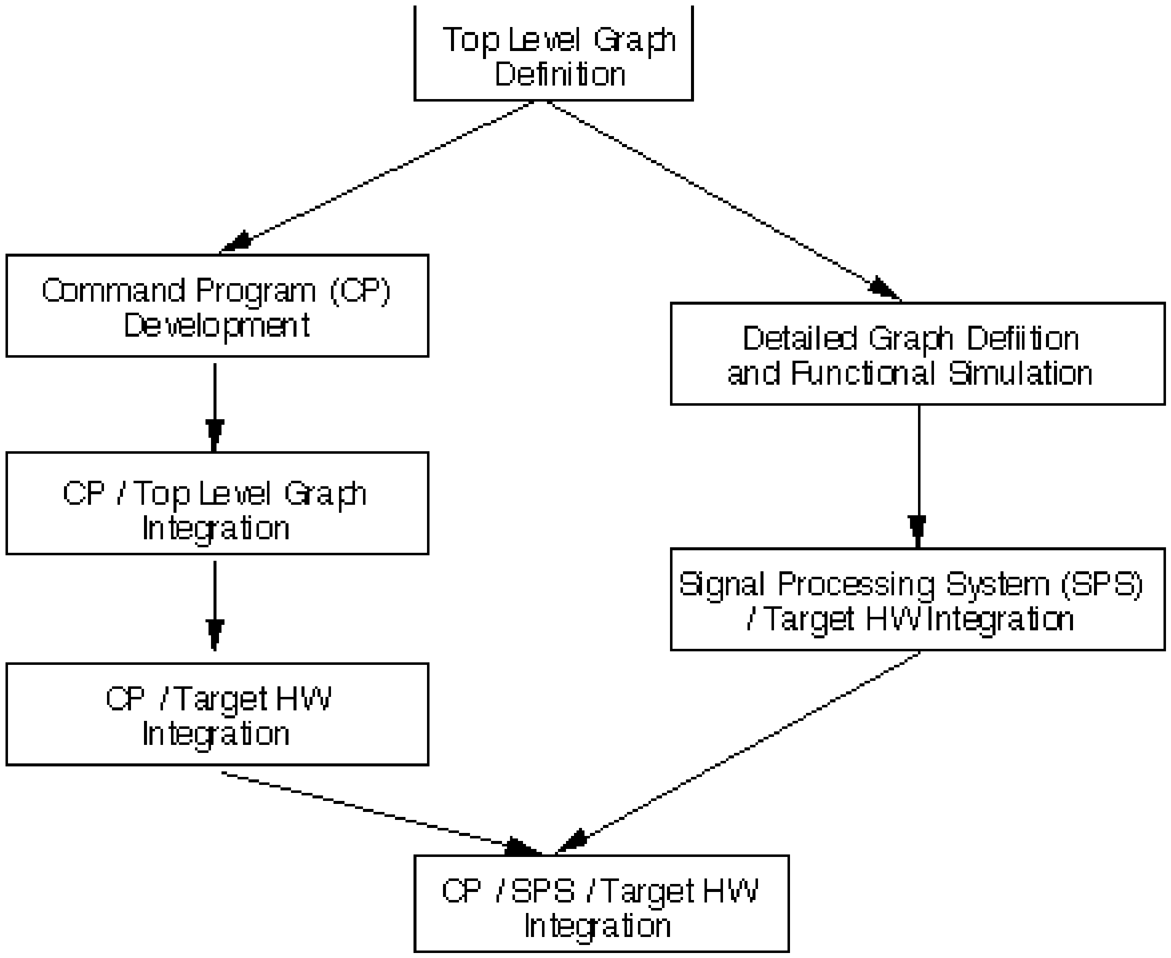

Top Level Graph Definition
Command Program (CP) Development
Detailed Graph Defiition and Functional Simulation
CP / Top Level Graph Integration
Signal Processing System (SPS) / Target HW Integration
CP / Target HW Integration
CP / SPS / Target HW Integration

Figure 3 - 5: System Development Process

After the top level graphs are captured, command program development proceeds. Following CP development, the CP is integrated with the top level graph. The Top Level graph at this stage only needs to perform data flow adequate to exercise the command program. Since graphs can easily be retargeted in GEDAE™ this integration can be performed on the CP development host. The next step is to verify that the actual command program executes as expected with the target hardware and in cooperation with the rest of the command and control system. This can also be performed independently from the detailed signal processing. The approach is to again use the top level graph but to retarget its execution to the final target hardware; this is easy to do in the GEDAE™ development environment. Concurrent with the development of the Command Program and its integration with the top level graphs, the DSP application graphs are being completed and tested on the target hardware. When the signal processing graphs are completed and tested, the final system integration uniting the command program and the final signal processing graphs may occur.

After the top level graphs are captured, command program development proceeds using Verilog's ObjectGEODE tool. The model is simulated with the interface to the actual data flow graphs implemented as a set of smart stubs. A smart stub is a software program which simulates simple responses from the signal processing system. Command program functionality can thus be verified independent from the implementation of the signal processing. Concurrent with the command program development is the detailed definition of the application graphs in GEDAE™. The top level graphs are detailed and simulated using GEDAE™ to establish correct funtionality.

3.4.1.5 Target Software Generation and Test

When the candidate architecture is assembled, the autocoding capability of the signal processing and control processing development tools is used to generate the final software for the target architecture. The same visualization capability available during algorithm development and simulation is maintained through software debugging on the target hardware. The integrated RASSP architecture toolset provides the same user interface throughout the development process from concept to final software generation.

3.4.2 Integrated Toolset

As part of the RASSP development ATL has looked at a wide range of tools applicable to the HW/SW Codesign activity. These developments have resulted in the Integrated RASSP Archtecture Toolset. The intent of this toolset is to facilitate HW/SW Codesign through an integrated capability for specifying the signal processing application and its control, performing architecture tradeoffs using VHDL token-based performance models, defining the required functionality in the form of data flow graphs and control states, simulating the associated functionality of the data and control flow, and autocoding the final software (both signal processing and control) for the target hardware. These functions are performed using the GEDAE™, PMW (recently renamed Cosmos), and ObjectGEODE. Part of the RASSP effort has involved both enhancing and integrating these tools as shown in Figure 3 - 6.

The assumption in the discussion that follows is that the hardware and software models required to construct an architecture and/or signal processing graph exist in a reusable library. Although this assumption is made for the purposes of discussion, many of the graph primitives used in the application were generated specifically for this application. In general, new primitives will often be required and consequently tools must support easy primitive generation and insertion into the system. In addition to the three tools shown in Figure 3 - 6, ATL has developed an Application Specific Interface Builder (AIB) which is also discussed. Its purpose is to provide a more natural interface between the signal and control processing. The example used in the following discussion is the SAIP (Semi-Automated IMINT Processor) application which was the 4th RASSP Benchmark.

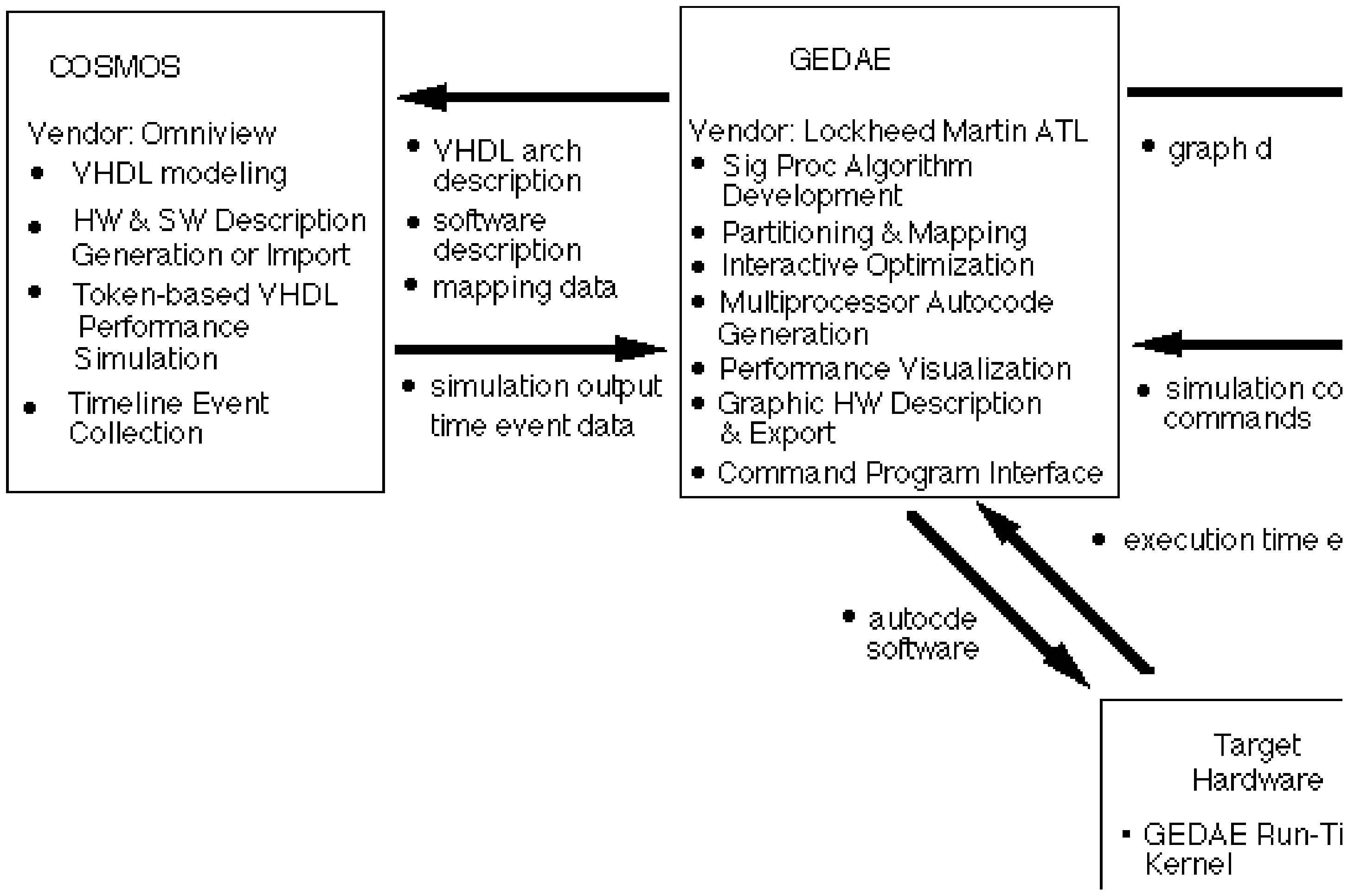

COSMOS
Vendor: Omniview
• VHDL modeling
• HW & SW Description Generation or Import
• Token-based VHDL Performance Simulation
• Timeline Event Collection
• VHDL arch description
• software description
• mapping data
• simulation output time event data
GEDAE
Vendor: Lockheed Martin ATL
• Sig Proc Algorithm Development
• Partitioning & Mapping
• Interactive Optimization
• Multiprocessor Autocode Generation
• Performance Visualization
• Graphic HW Description & Export
• Command Program Interface
• graph d
• simulation co commands
• execution time e
• autocde software
Target Hardware
• GEDAE Run-Ti Kernel

Figure 3 - 6: RASSP Architecture Toolset

3.4.2.1 GEDAE™ - An Algorithm Development/Autocoding Tool

This section briefly describes the GEDAE™ software, one of the tools in the HW/SW Codesign Toolset. Lockheed Martin ATL has been developing GEDAE™ for over a decade. GEDAE™ supports graphical programming and autocoding of applications for execution on workstations and embedded hardware. Under the RASSP program, GEDAE™ has been enhanced and integrated with the other tools in the HW/SW Codesign Toolset.

Overview - GEDAE™ is a highly interactive graphical programming and autocoding environment which facilitates application development, debugging, and optimization on workstations or embedded systems. It's graphical editor supports building data flow graphs which are very readable. Explicit inputs and outputs are identified and user notes can be inserted directly on the graph canvas. The same user interface which supports the graphical editor is used for controlling all activity within GEDAE™. It is not necessary to switch tools or become familiar with a different user interface when moving from algorithm development to embedded code generation which minimizes the learning curve and reduces tool training. The interface has been aclaimed as highly intuitive by observers at conferences and the visualization is unmatched for analizing system solutions. The overall philosophy employed in the development of GEDAE™ is "never take intelligent decisions out of the hands of the designer - but rather provide the tools and functionality needed to improve productivity and decision making, automating as much of the drudgery as possible". Capability is provided for the designer to readily partition graphs and map the partitions to multiple workstations or multiple processors in an embedded system. Autocoding generates appropriate schedules and code for each processing element which is efficient in terms of execution time and memory usage.

A GEDAE™ Run-Time Kernel provides all of the interprocessor communication required by the particular software mapping. Although the designer has flexibility in selecting the type of communication used (e.g. socket, DMA, or shared memory), implementation of the communication is automatic. Therefore, the application developer never needs to write any interprocessor communication software. This may in fact be the largest benefit of graph based programming for multiprocessors since our experience indicates this area to be responsible for most multiprocessor system debugging problems.

Algorithm Capture - Algorithms are captured in GEDAE™ by placing processing functions extracted from a library on a canvas and interconnecting them using the extensive facilities of the graphical editor. Using a top down design approach, the basic building blocks and data passing requirements can be put on the canvas. In this way, the application designer can build, test, and analyze algorithms with point and click simplicity. Designers can select from functions contained in the extensible library. Library functions are provided for most of the commonly used signal processing functions. Understanding that a function library will never be complete, templates are provided for creating new functions. In addition to providing all of the typical data types, GEDAE™ has the important capability to define new arbitrary data types(e.g. complex C structures) for use with custom primitives. This ability is of great importance to users who want to capture heritage software which is generated in modules whose I/O is maintained as complex data structures.

Hierarchy is supported in a flow graph which simplifies complex application understanding. A typical hierarchical graph is shown in Figure 3 - 7 with the hierarchical function titled "range_p" expanded. The application is Synthetic Aperture Radar(SAR) image generation which was used as a benchmark on the RASSP program. All functions in the top level graph are hierarchical and can be expanded by double clicking on the box title bar. Unlimited nested hierarchy is supported.

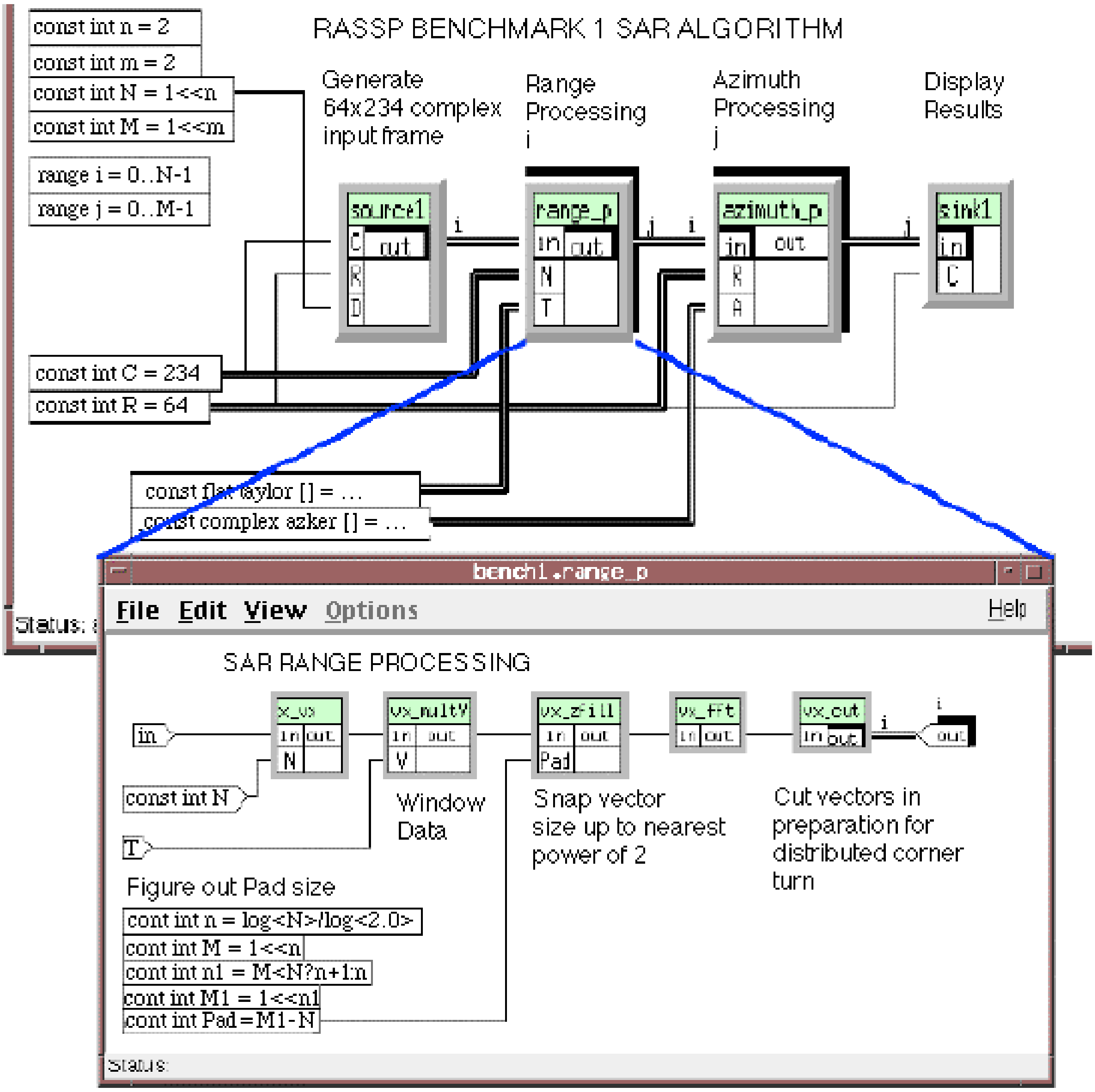


Figure 3 - 7: Example SAR application graph generated using the GEDAE™ software

GEDAE™ supports extensive and efficient parallel processing. Parallelism is succinctly represented in the SAR graph. The bold shadow surrounding some of the graph nodes as well as some of the node inputs and outputs indicates a 'family' of functions and the double lines between nodes indicate a family of interconnects. The indices above any family function (e.g. "i" above the range_p box) indicates that there are "i" of these functions. The output of the source1 box is a family of "i" outputs for processing by the "i" range_p functions.

The famiily of outputs is represented by the shadow around the source1 "out" label and the label "i" on the arc. In addition the output of each range processing family member is also a family which indicates that each section of data processed by one of the range processing functions is output as a family of outputs. When embedding this application, the individual family members are mapped to different processors for concurrent processing. Of particular note is that the alternating indices between the output of range processing and the input to azimuth processing specifies a distributed corner turn since the "j-th" output from the "i-th" range processor is routed to the "i-th" input of the "j-th" azimuth processor.

When the interconnection of the links between family members becomes complex, special routing boxes can be inserted to define the connections. These routing boxes are merely a graphical representation aid and do not consume processing resources.

Simulation / Validation - Execution of GEDAE™ graphs is controlled through the same interface used to construct the graph. Users can modify parameters in the graph on the fly and observe t2he impact of those changes. The ease of making modifications to a graph and its operating parameters increases productivity by making it easy for designers to fine tune applications quickly. Execution results are presented to the designer in the form of detailed timelines and execution schedules along with memory maps to support the designers analysis of system solutions. An annotated example of a trace table is shown in Figure 3 - 8. It contains both a hardware execution profile and a software execution profile. Computation time, data flow activity(buffers filling and emptying), and communication activity are all detailed in the trace table.

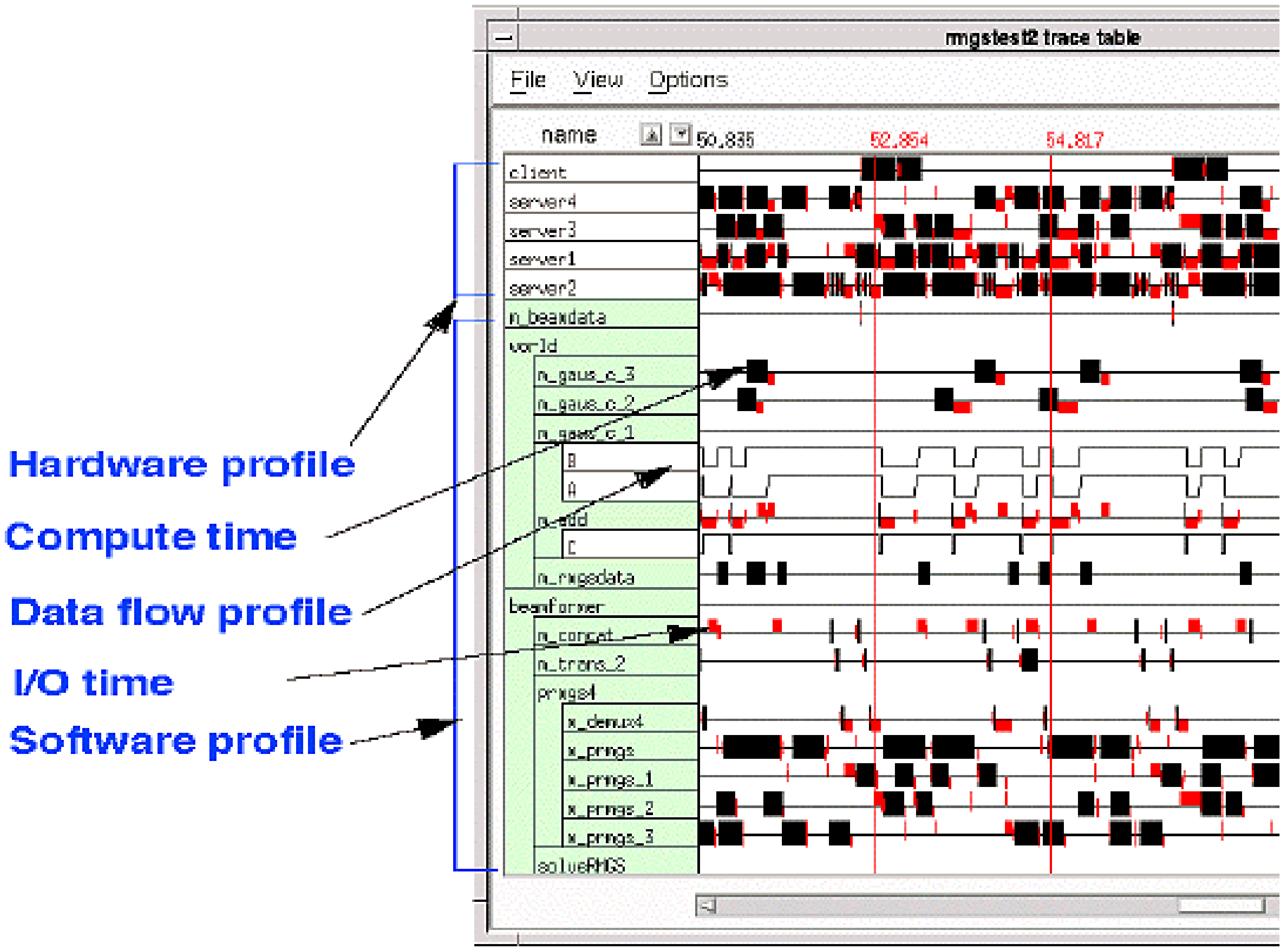
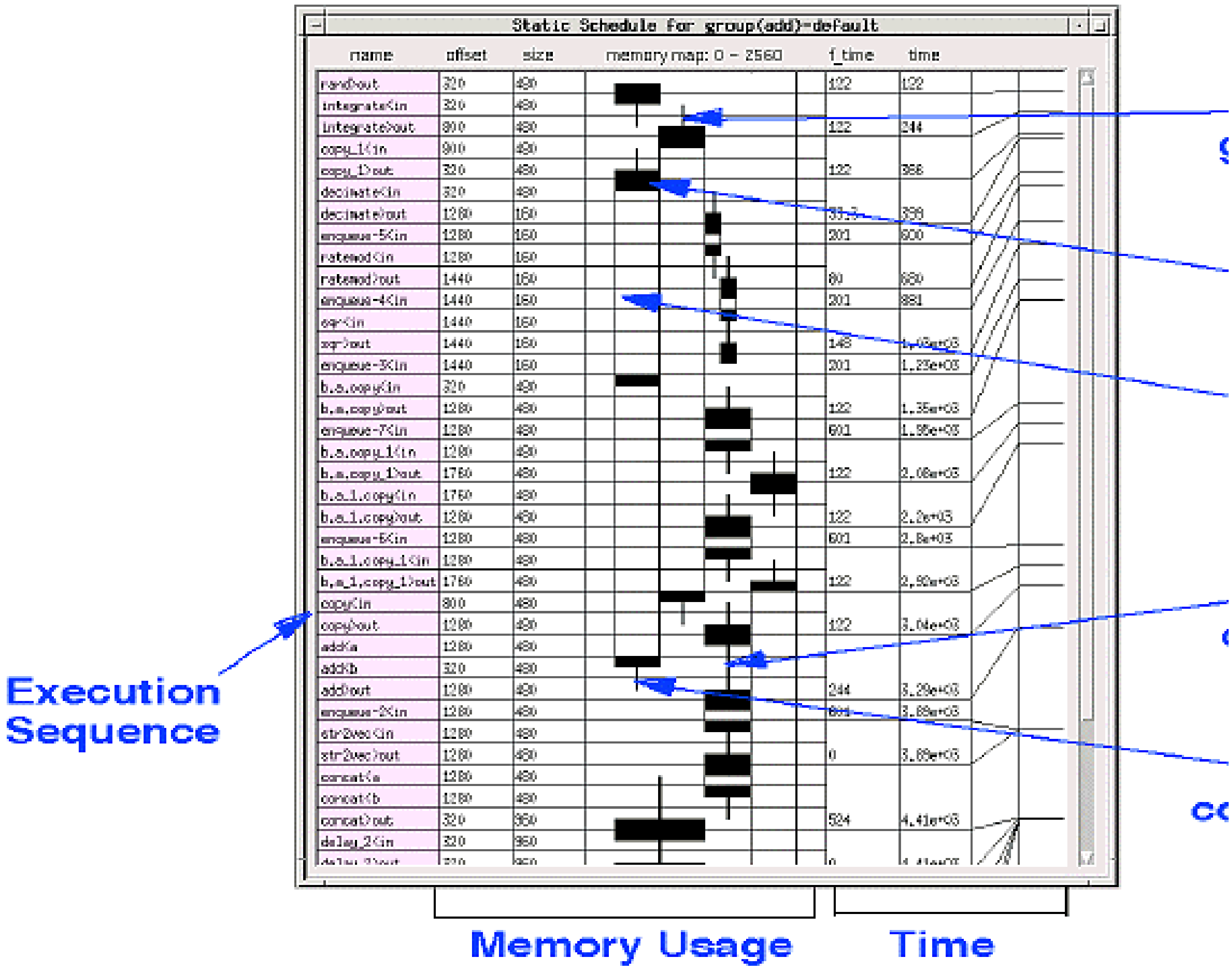


Figure 3 - 8: Detailed timeline data for system analysis

Embedded Code Generation - GEDAE™ provides an efficient autocoding capability driven by partitioning and mapping defined by the user. Because GEDAE™ handles all interprocessor communication, the designer never has to write any communication software. GEDAE™ launches the compilation, linking, loading and execution of the application on the embedded hardware. An embedded run-time kernel on each processor supports execution. GEDAE™ generates the execution schedule for each processor and provides the user the ability to divide schedules into sub-schedules which may all operate at different firing granularity to optimize performance. Execution schedules and memory maps are presented for analysis as shown in Figure 3 - 9.



Static Schedule for group(add)-default

name	offset	size	memory map: 0 - 2560	f_time	time
rand>out	320	480		122	122
integrate<in	320	480			
integrate>out	800	480		122	244
copy_1<in	800	480			
copy_1>out	320	480		122	366
decimate<in	320	480			
decimate>out	1280	160		33.3	399
enqueue-5<in	1280	160		201	600
ratemod<in	1280	160			
ratemod>out	1440	160		80	680
enqueue-4<in	1440	160		201	881
sqr<in	1440	160			
sqr>out	1440	160		148	[illegible]
enqueue-3<in	1440	160		201	1.23e+03
b.a.copy<in	320	480			
b.a.copy>out	1280	480		122	1.35e+03
enqueue-7<in	1280	480		601	1.95e+03
b.a.copy_1<in	1280	480			
b.a.copy_1>out	1760	480		122	2.08e+03
b.a_1.copy<in	1760	480			
b.a_1.copy>out	1280	480		122	2.2e+03
enqueue-6<in	1280	480		601	2.8e+03
b.a_1.copy_1<in	1280	480			
b.a_1.copy_1>out	1760	480		122	2.92e+03
copy<in	800	480			
copy>out	1280	480		122	3.04e+03
add<a	1280	480			
add<b	320	480			
add>out	1280	480		244	3.29e+03
enqueue-2<in	1280	480		601	3.89e+03
str2vec<in	1280	480			
str2vec>out	1280	480		0	3.89e+03
concat<a	1280	480			
concat<b	1280	480			
concat>out	320	960		524	4.41e+03
delay_2<in	320	960			
delay_2>out	320	960		0	4.41e+03

Figure 3 - 9: GEDAE™ software execution schedule and memory map display

Optimization - GEDAE™ supports the optimization of partitioning and mapping, memory usage, communication mechanism selection, schedule firing granularity, queue capacities, and scheduling parameters. GEDAE™ provides the ability to interactively manipulate these items which greatly improves the designers ability to optimize processing after retargeting an application to a new architecture.

3.4.2.2 Omniview Cosmos™ - A Performance Modeling Tool

Cosmos from Omniview Design Inc. is a performance modeling environment that allows the user to quickly obtain data about a system's performance, such as latency and throughput. Both hardware and software components of a system can be modeled simultaneously. Cosmos is intended to be used as a high-level system design tool, where early design trade-offs in software/hardware partitioning and architecture are made. Currently, such trade-offs (such as the networking architecture to use, the number, type and speed of the processors to use etc.) are done on the basis of previous experience, spreadsheet based calculations etc. However, these calculations may not accurately capture the dynamics of the system, or may be insufficient to give the user an accurate prediction of the performance of a given system. Using a simulation based performance modeling environment such as Cosmos allows the user to obtain the performance characteristics of different system architectures early in the design cycle, thus allowing the selection of the best architecture amongst various alternatives before detailed design is begun. This potentially saves a lot of re-design and the associated costs.

Using Cosmos, the user builds a model of a system. A model consists of hardware elements connected together into the schematic of the system, software tasks that model the software components of the system, and a mapping that defines which software tasks run on which processors in the system. After a model is built, it is simulated using a commercial VHDL simulator. The simulation results are then imported back into Cosmos to be analyzed. Cosmos includes a complete set of analysis tools that allow the user to look at the behavior of the system in detail, and identify potential problems such as network bottlenecks, insufficient processing power etc. Cosmos provides a complete set of versioning and data storage features so that different versions of a system model can be saved and compared to each other for choosing the best version.

Cosmos Usage - Cosmos models systems using a VHDL based library of elements such as processors, networking elements etc. The models are un-interpreted, token-based models, in which actual data is not simulated, but only the interaction of elements based on the size and other characteristics of the data is simulated. This allows a very large increase in simulation times (by a factor of as much as 1000 over full-functional simulation), and also allows large systems (consisting of tens or hundreds of processors) to be simulated easily.

The Cosmos user builds a model of a system using the library elements for the hardware description. This is done using a built-in schematic editor. The hardware elements in the library include various commercial processors, networking elements such as VME, Mercury Raceway and Myrinet, and other data processing elements such as memories, disks, and data generators and data sinks. The various elements are instantiated into a system model and each instance can be customized by changing the values of parameters such as latency, throughout and other element-specific parameters. The elements are connected together in the schematic editor to define the hardware architecture of the model.

The software description is built using flow-charts, each of which represents a task running on one or more of the processors in the system. Each processor element in the library supports a real-time multi-tasking OS model, which can be customized to simulate the performance characteristics of different operating systems such as MCOS and VxWorks etc. The OS model also supports built-in inter-task communication mechanisms that allow different tasks to communicate with each other. A task in the software description of the model can simulate execution of instructions by using built-in mechanisms of the processor model. This allows the performance of the task to be modeled. The task is independent of the processor it runs on, so it is possible to model the performance of the same task(s) on different processors to understand the impact of changing processor architectures.

The tasks defined in the software model are mapped to the various processors instantiated in the hardware model using a mapping editor. This allows the user to easily change the mapping of software to hardware in the system model, without changing the hardware or software models.

Once the system hardware, software and mapping are defined, Cosmos generates a VHDL model of the system. This model uses a Cosmos supplied, VHDL based library as a basis for simulation of the system. The simulation is performed using a commercial VHDL simulator (Cosmos supports all the leading VHDL simulators). The simulation results are imported back into Cosmos to be analyzed, using the Cosmos analysis tools.

The simulation results can be analyzed using various tools that allow the user to examine the simulation as a whole, or examine the activity in the system at a given instant of time. The simulation can be played and/or stepped through forward or backward to examine events in sequence. The analysis tools include the Activity TimeLine, the HotSpot Analyzer, the Performance Metric Analyzer, and the Histogram Tool. The Activity TimeLine displays the entire simulation as a plot of time vs. activity of each individual hardware instance and software task. The TimeLine display shows overall activity of the system, and any specific activity can be examined in detail to indicate what each element and task was doing at any given time. The HotSpot Analyzer shows the utilization of the hardware elements in the system at any given instant of time. As the simulation is played back and forth, the HotSpot shows element utilization on a color-temperature scale. This makes it easy to detect over-utilized or under-utilized elements in the system. The Performance Metric Analyzer (PMA) shows the instantaneous values of various performance metrics such as utilization, latency and throughput for any set of elements in the system. The Histogram Tool allows the user to plot these same metrics for the simulation as a whole, giving a picture of the performance of an element over the entire simulation.

Figure 3 - 10 shows the relationship of the various elements of Cosmos.

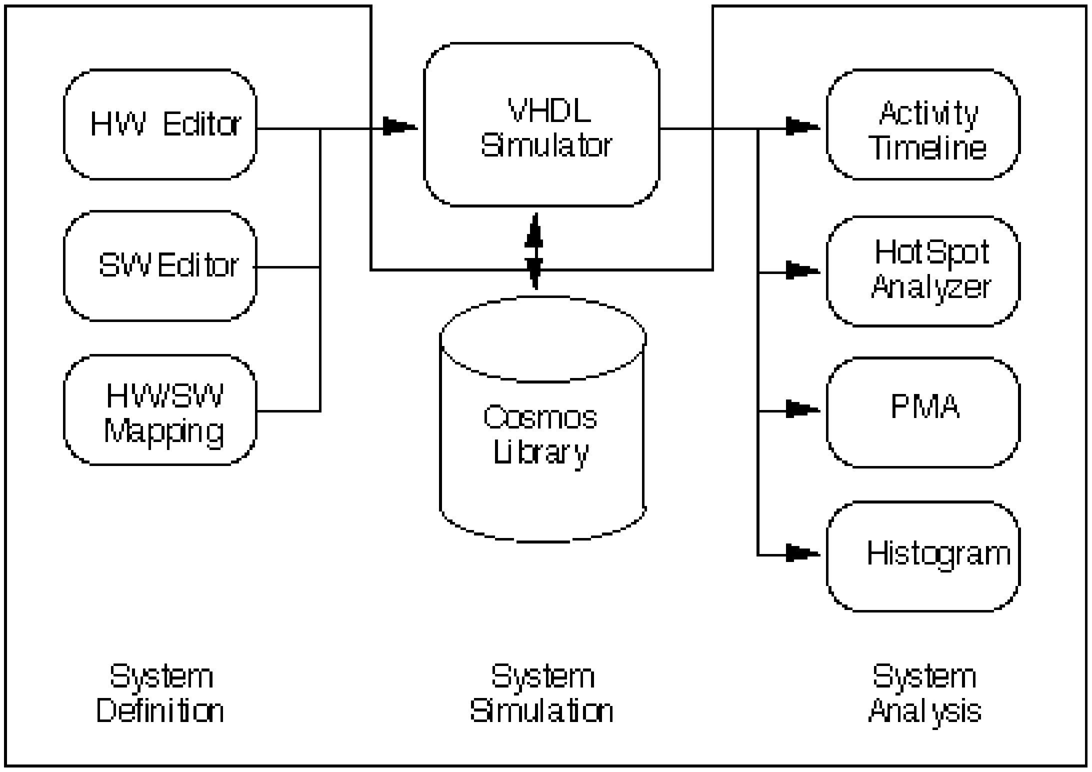


Figure 3 - 10: Elements of Cosmos

3.4.2.3 ObjectGEODE - A Control Software Development and Autocoding Tool

ObjectGEODE from Verilog SA is a toolset dedicated to analysis, design, verification and validation through simulation, code generation and testing of real-time and distributed applications that are best represented by a finite state machine model of computation. Such applications are used in many fields such as telecommunications, aerospace, defense, process control or medical systems.

ObjectGEODE supports a coherent integration of complementary approaches based on standards. These standards are:

- OMT (Object Modeling Technique), now UML (Unified Modeling Language),
- SDL (Specification and Description Language) issued from an international standard organization, ITU-T(former CCITT)
- MSC (Message Sequence Chart) also issued from ITU-T.

OMT/UML Class, Instance Diagrams and StateCharts are used to perform system requirements analysis. SDL includes Architecture Diagrams for system structure, Interconnection Diagrams for system communication and Extended Finite State Machine Diagrams for system behavior. MSC includes Message Sequence Charts for Use or Test case definition and Scenario Diagrams grouping Sequence Charts for function description. Data can be described with ISO's ASN.1. All these diagrams are closely interrelated thus removing any possible discontinuity right through to the final implementation stage.

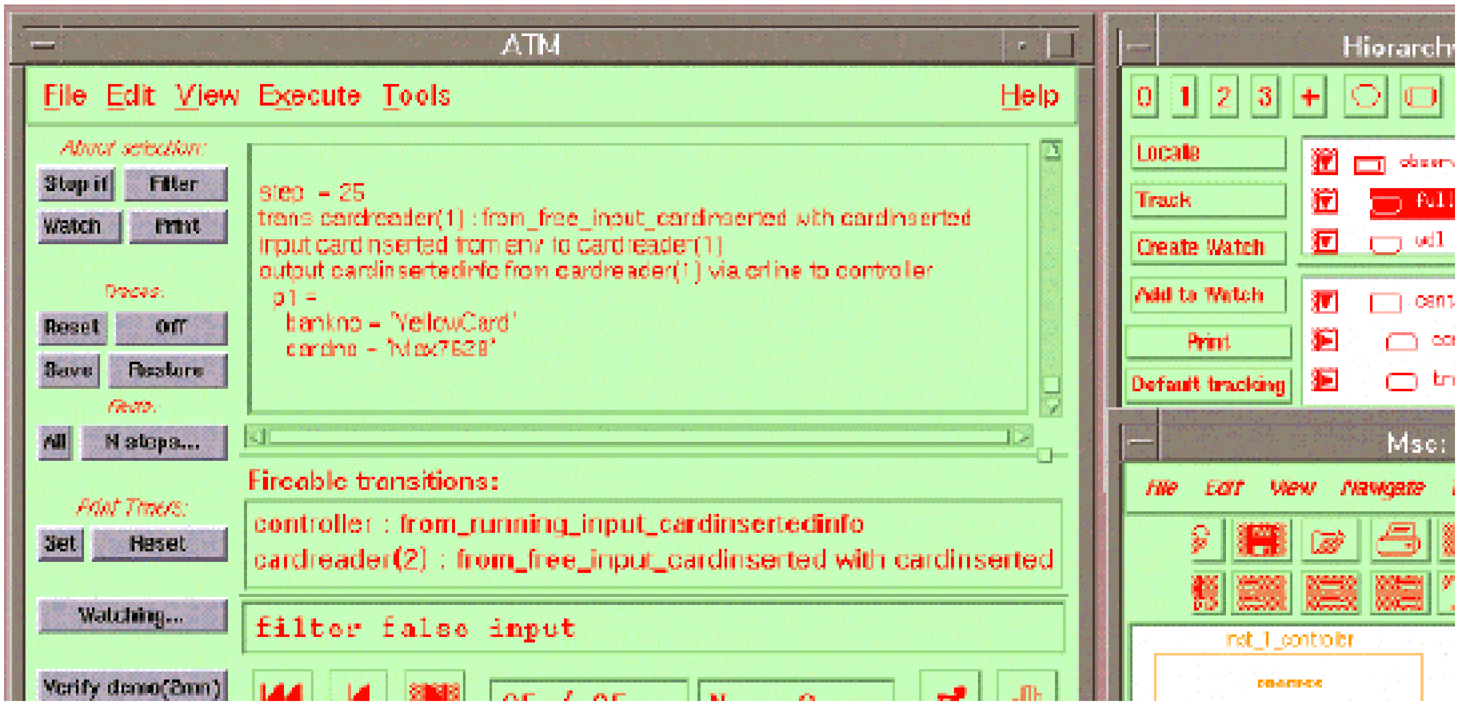
The OO approach implemented in ObjectGEODE, supports project members on a long-term basis and takes reuse into account at all levels.

ObjectGEODE features - ObjectGEODE provides the following features.

- Various editors provide for intuitive means of creating, modifying, and viewing the diagrams of an ObjectGEODE description: Class, Instance, Scenario, Architecture, Interconnection and Message Sequence diagrams. The Checker controls consistency and compliance with notation rules. Graphical representation can be generated to professional documentation standards. Powerful multi-user features are available for large/distributed projects.
- A powerful simulation and formal verification and validation tool is provided to graphically detect before coding start, any pathological behavior pattern or show proof that the ObjectGEODE model complies with requirements.
- Fully executable C and C++ code of the (distributed) multi-task real-time application is automatically generated. Makefiles are also generated to automate the building process. The generated code maps the production real-time executive (such as VxWorks, pSOS+ or VRTX), and network protocols (such as TCP/IP) via a dedicated ObjectGEODE Run-Time Library. The DesignTracer allows the current design description to be visualized interactively and trace information (including time) to be displayed as Message Sequence Chart diagrams. Test sequences can be generated in the ISO standard Table and Tabular Combined Notation (TTCN) format, and
- Integrated into Configuration Management Systems and development environments.

Figure 3 - 11 shows a typical screen from an ObjectGEODE simulation session.

Host and Target Systems Supported - The ObjectGEODE toolset runs on SUN, HP, IBM RS/6000, DEC Alpha workstations and on Windows NT. Most popular target systems such as VRTXsa, pSOS+, VxWorks,or UNIX are supported as well as TCP/IP for distributed communications.



```
step  = 25
trans cardreader(1) : from_free_input_cardinserted with cardinserted
input cardinserted from env to cardreader(1)
output cardinsertedinfo from cardreader(1) via cdline to controler
  p1 =
    bankno = 'YellowCard'
    cardno = 'Max7628'
```

Fireable transitions:

```
controller : from_running_input_cardinsertedinfo
cardreader(2) : from_free_input_cardinserted with cardinserted
```

```
filter false input
```

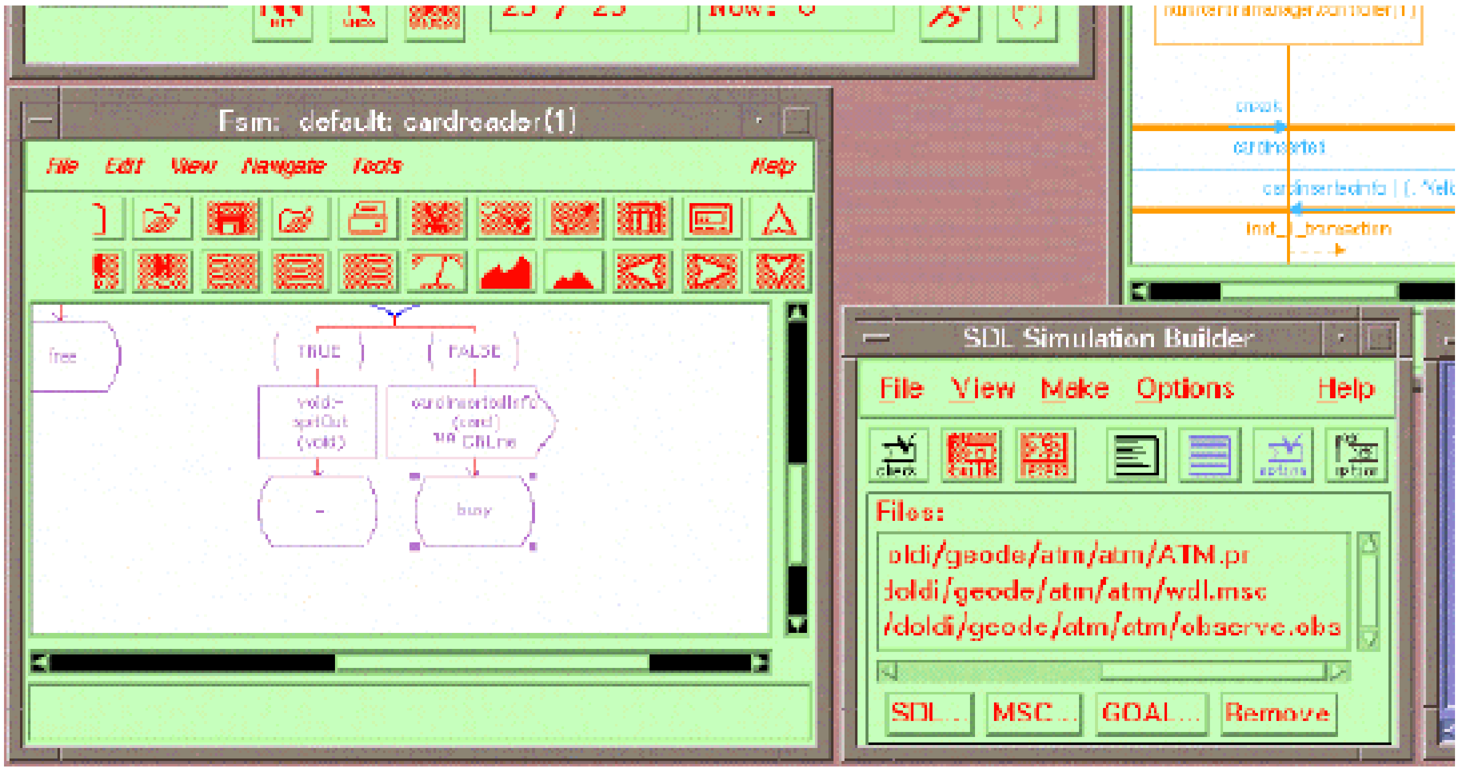

Fsm: default: cardreader(1)
File Edit View Navigate Tools Help
free
TRUE
FALSE
void:=
splitOut
(void)
cardinsertedinfo
(card)
CBLine
-
busy
SDL Simulation Builder
File View Make Options Help
Files:
oldi/geode/atm/atm/ATM.pr
/oldi/geode/atm/atm/wdl.msc
/oldi/geode/atm/atm/observe.obs
SDL...
MSC...
GOAL...
Remove
cardinserted
cardinsertedinfo
init_transaction

Figure 3 - 11: An ObjectGEODE Simulation Session

3.4.2.4. Application Interface Builder

As shown in Figure 3 - 12, the command program views the signal processing system as a collection of modes and submodes, while the signal processing system naturally presents itself as a collection of graphs. Consequently two interfaces are defined. The Command Program Interface (CPI) provides control services built around the signal processing graphical notions. The Application Specific Interface (ASI) provides a view of the signal processing as a collection of modes and submodes. The ASI is a more natural representation for system designers while the CPI is natural to the data flow graph designers. As Figure 3 - 12 illustrates, there is a gap between the two interfaces. An Application Interface Builder (AIB) has been developed by ATL under the RASSP program to generate the Application Specific Interface that fills this gap. The control software developer issues a mode change request via the ASI and the interface software constructed by the AIB will provide the detailed sequence of commands via the CPI to implement the request. The ASI instantiation consists of calls to the CPI based on the specific set of modes / submodes for the application, the set of graphs developed to perform the application, and the correlation between the two sets. The capability of the AIB will be commercialized as part of the GEDAE™ product.

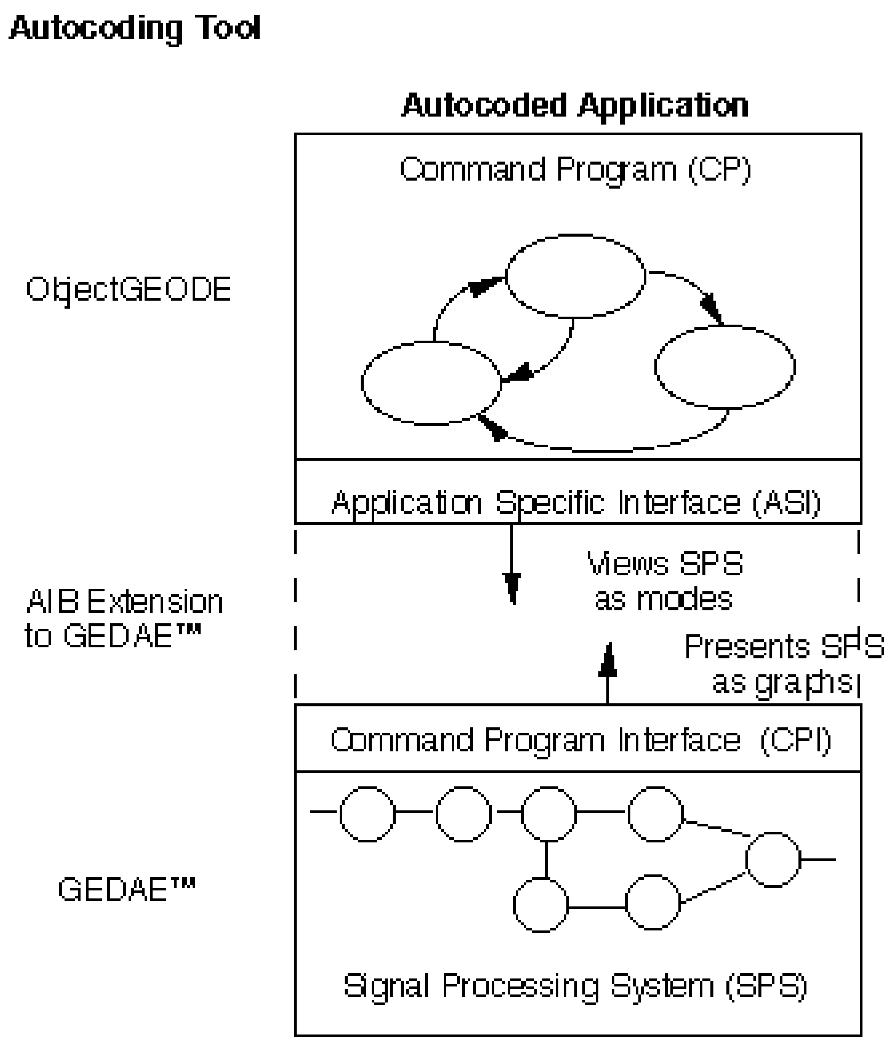


Figure 3 - 12: Application Interface Builder bridges gap between data flow and control flow

The overall command program is a layered architecture as shown in Figure 3 - 13. The bottom two layers address the data flow graph perspective while the top two layers address the control perspective. The bottom most layer is the Command Program Interface (CPI) layer and consists of the generic set of graph control functions provided by GEDAE™. This layer can be viewed as Commercial of the Shelf (COTS) software from the perspective of the command program. The next layer is the Application Specific Interface (ASI) layer. It is generated by the Application Interface Builder (AIB) and provides a high level set of functions used to instantiate, control, and configure a top level graph(s). The next layer up is the Executive Finite State Machine layer that reflects the state of the signal processing application. This layer may reflect a wide range of complexity and may be generated by hand or may require the utilization of a graphical tool such as ObjectGEODE. The User Interface layer may represent a simple interface for testing or may represent a complete interface for the application.

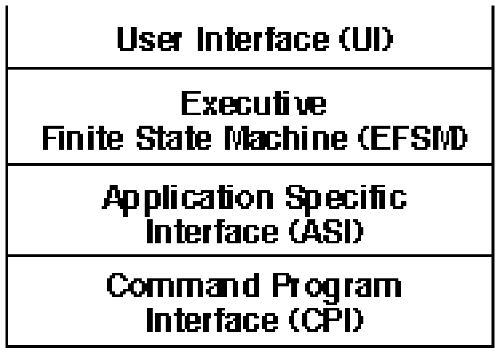


Figure 3 - 13: Layered Command Program Architecture

Approved for Public Release; Distribution Unlimited *Dennis Basara*

Next Up Previous Contents

Next: 5 Hardware / Software Codesign Process Applied to Mixed COTS / Custom Architecture **Up:** Appnotes Index **Previous:**3 RASSP Hardware / Software Codesign Process

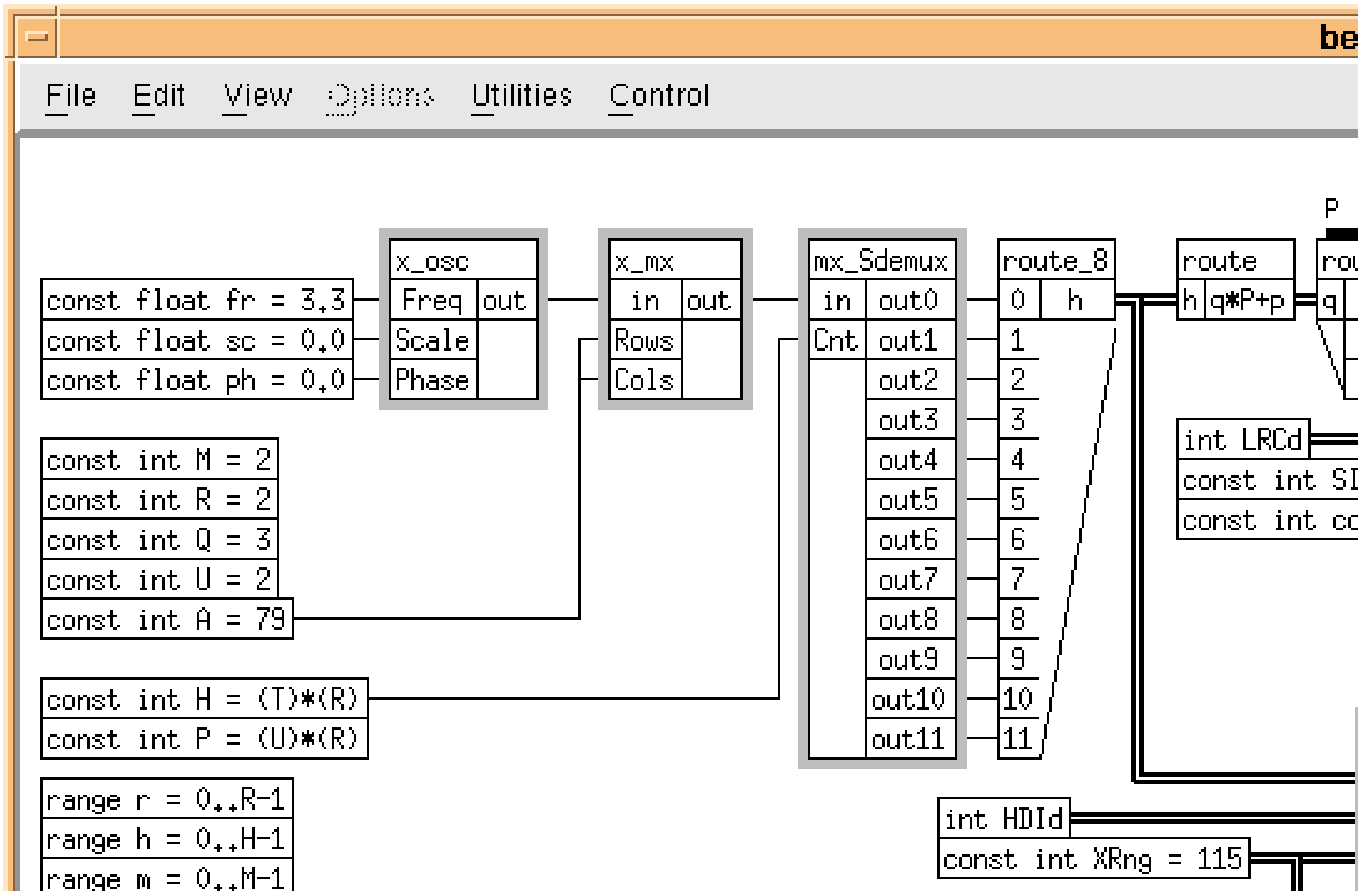
RASSP Hardware / Software Codesign Application Note

4.0 Integrated Toolset Example for COTS Architecture

The discussion that follows describes the application of the integrated toolset to the Semi- Automated IMINT Processor (SAIP) application which was implemented as the RASSP Benchmark 4. This benchmark has been implemented using a fully COTS archtecture.

4.1 Data Processing by Slices

A data flow graph constructed for the SAIP application is shown in Figure 4 - 1. The graph contains all of the data flow required by the application but the processing which must be performed is represented with simple delays. There is no functionality contained within the processing functions referred to as HDI, LRC and HRC. The graph was constructed using the graphical editor of the GEDAE™ tool. This graph is executable within GEDAE™, but because the processing is modeled as simple delays, no computation is performed on the data, it is merely moved throughout the graph to represent the actual data flow required by the application.



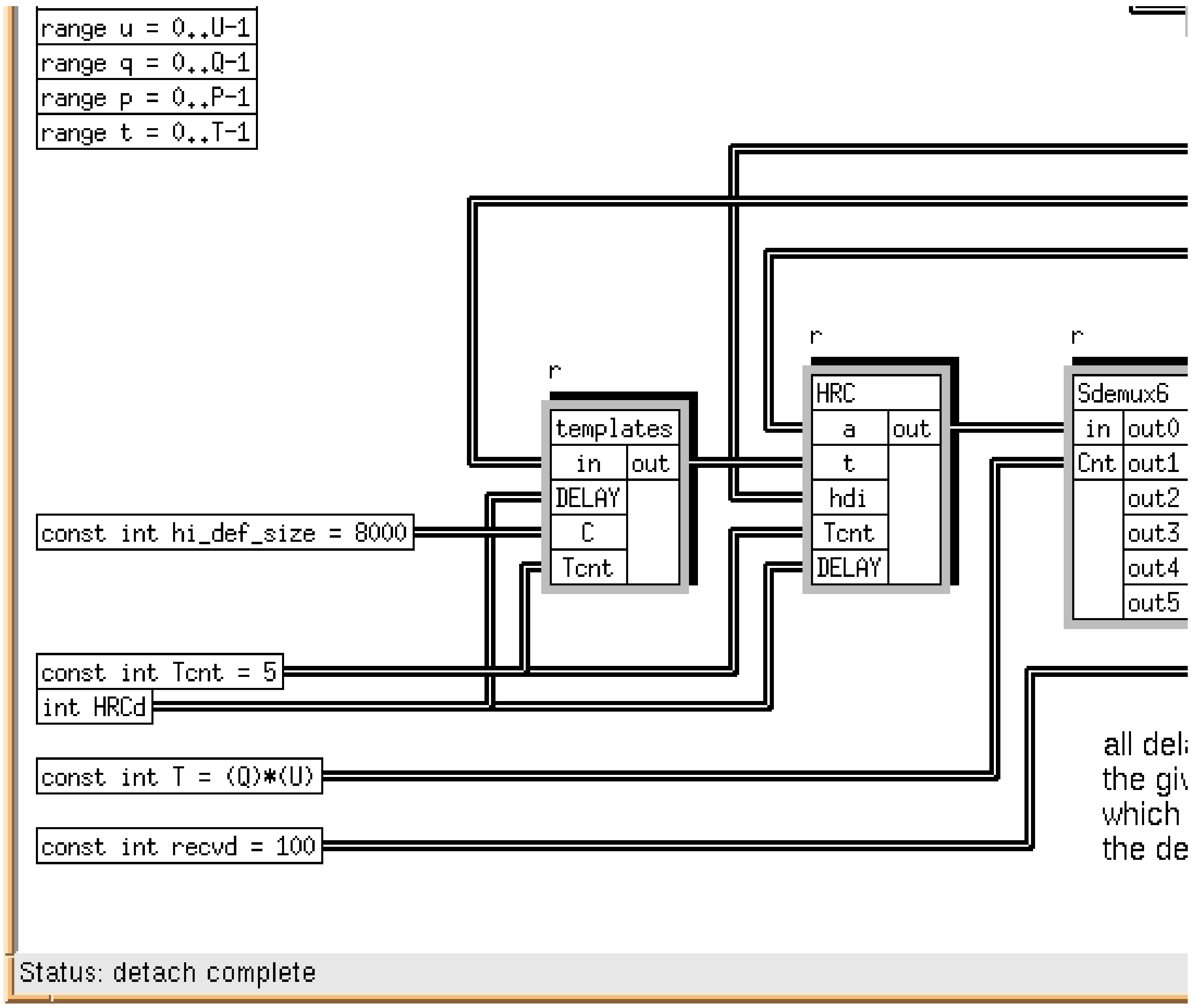
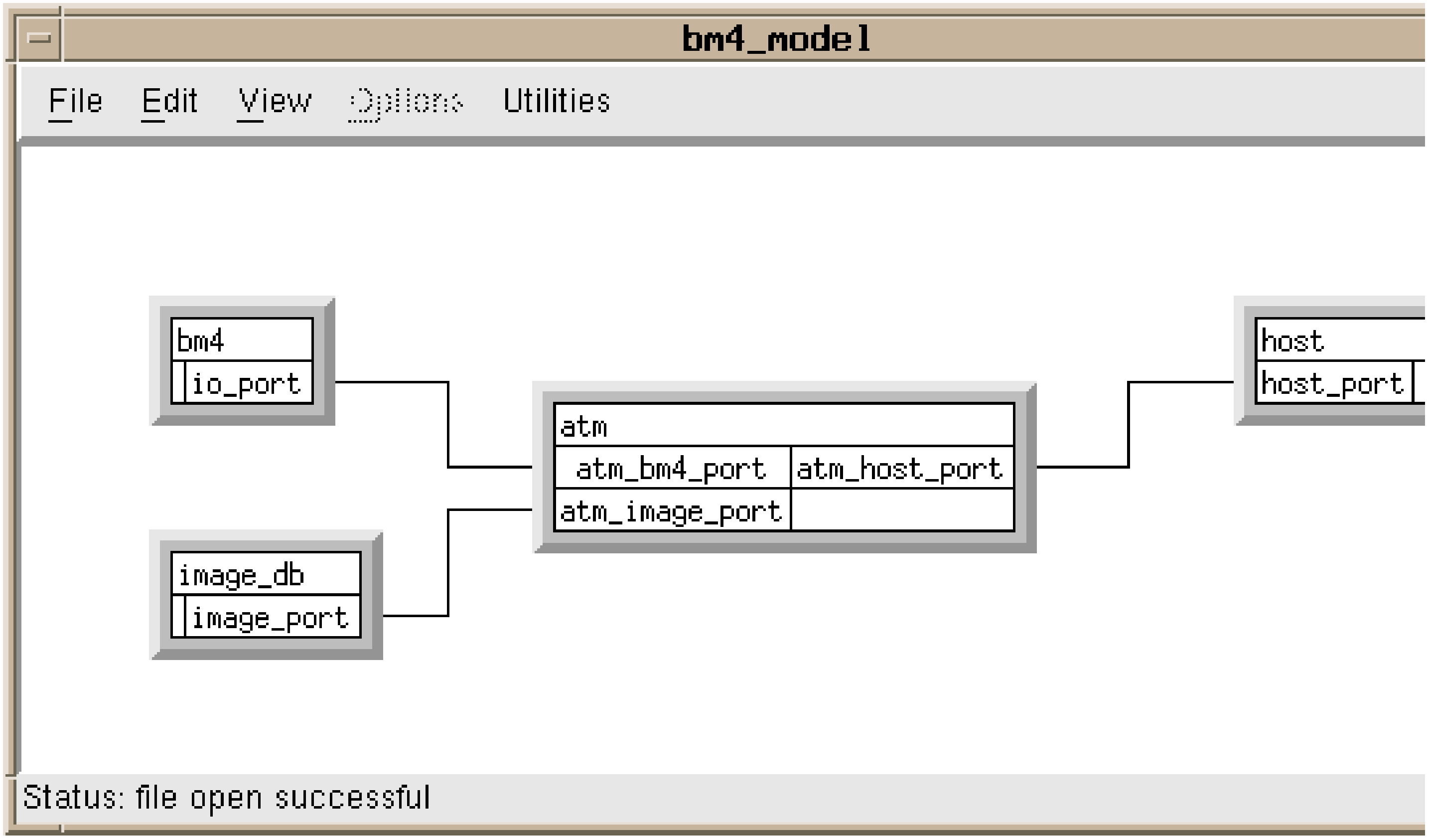


Figure 4 - 1: SAIP top level processing flow graph

4.2 Generating the hardware architecture

A portion of the hardware architecture for the SAIP application is shown in Figures 4 - 2 through 4 - 4. In Figure 4 - 2, the upper left window shows the top level architecture which consists of a host computer, ATM network, an image database, and the BM4 hardware. The lower left window is an expansion of the BM4 harware showing that it consists of a control board and two Alex boards (SHARC products from Alex Computer Systems). The expanded view of one of the Alex boards is shown in the right window of Figure 4 - 2 and is modeled as a VME bridge plus the remaining Alex hardware. Figure 4 - 3 shows the remaining Alex hardware modeled as two SHARC processors plus two SHARCPac daughterdcards. Finally, in Figure 4 - 4, one of the SHARCpacs is shown modeled as eight SHARC processors interconnected via SHARC links. The Alex board architecture utilizes only SHARC links for processor to processor communications, and all communication to/from the VME is through one of the SHARC processors on the motherboard. From these figures, it is seen that the architecture being modeled consists of two boards, each containing eighteen (18) SHARC processors. Four of these boards are required to meet the overall application throughput. All communication between processors is via the Sharc links. This two board partial architecture example was used to evaluate the mapping of the application to the individual processors on the board. The graphical, hierarchical description of the architecture shown in Figures 4 - 2 through 4 - 4 is generated using the GEDAE™ hardware editor and the individual elements correspond to architecture entitys in the COSMOS model library.



Status: file open successful

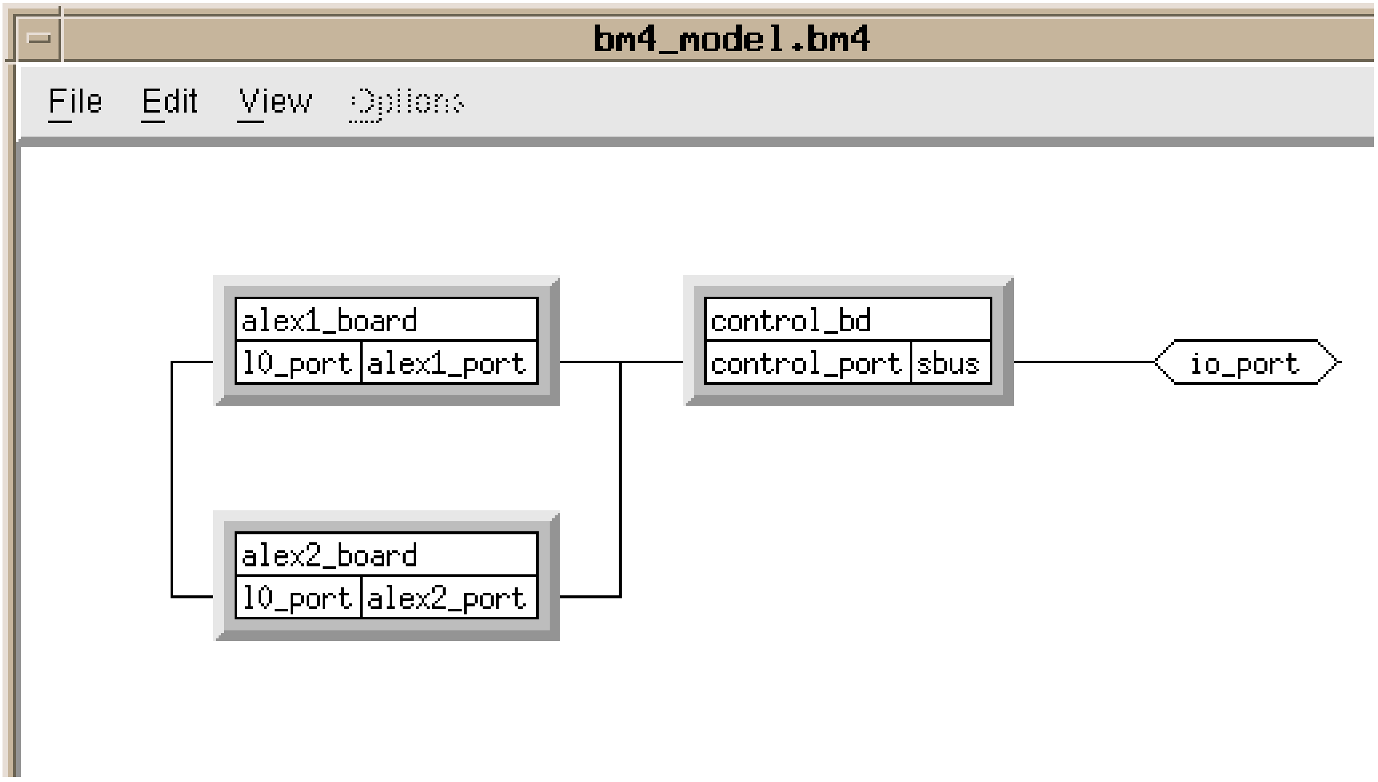


Figure 4 - 2: Graphical hardware model of the top level SAIP partial architecture, Benchmark 4 hardware, and top level view of Alex board

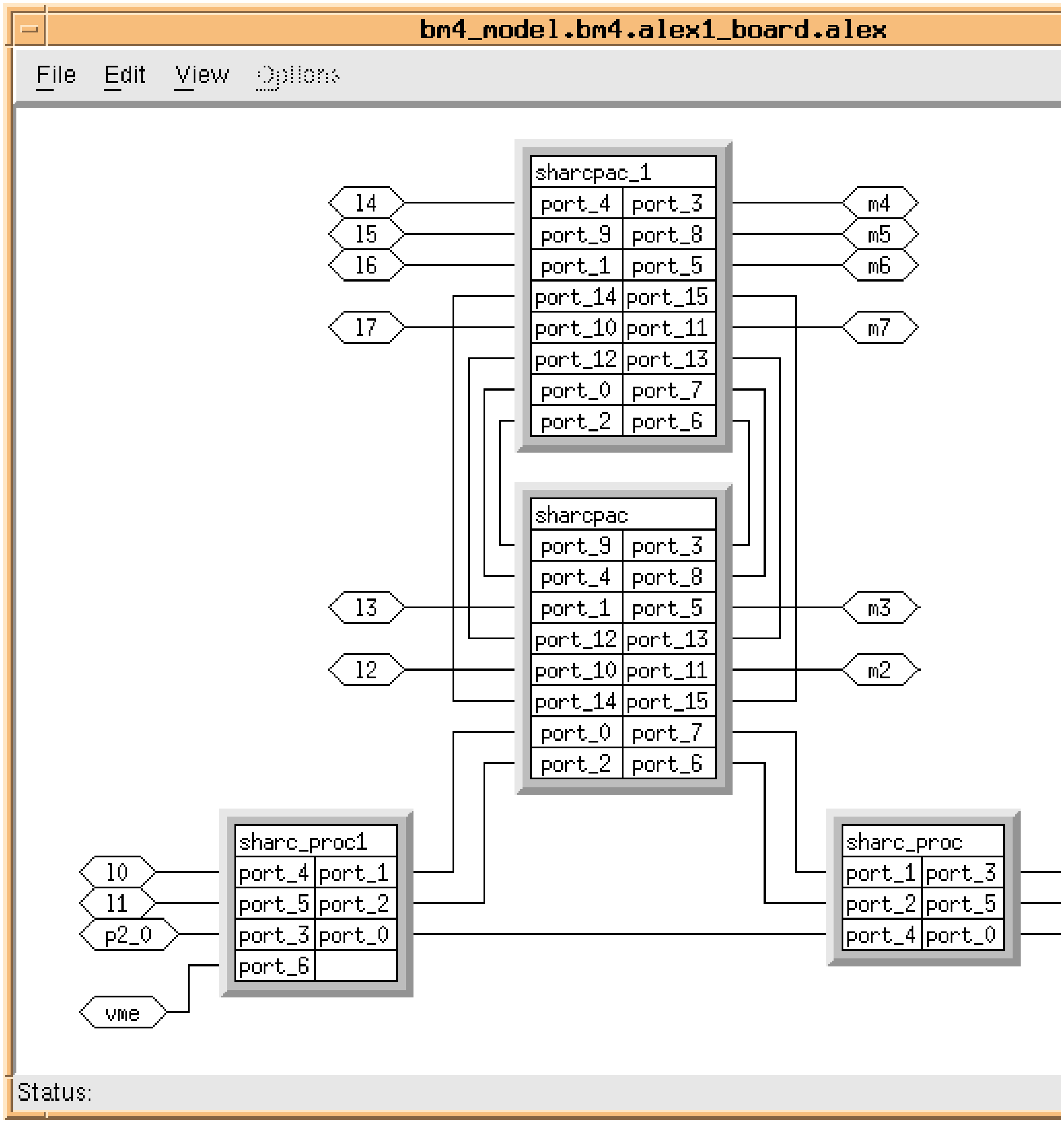


Figure 4 - 3: Graphical model of DSP hardware on Alex board.

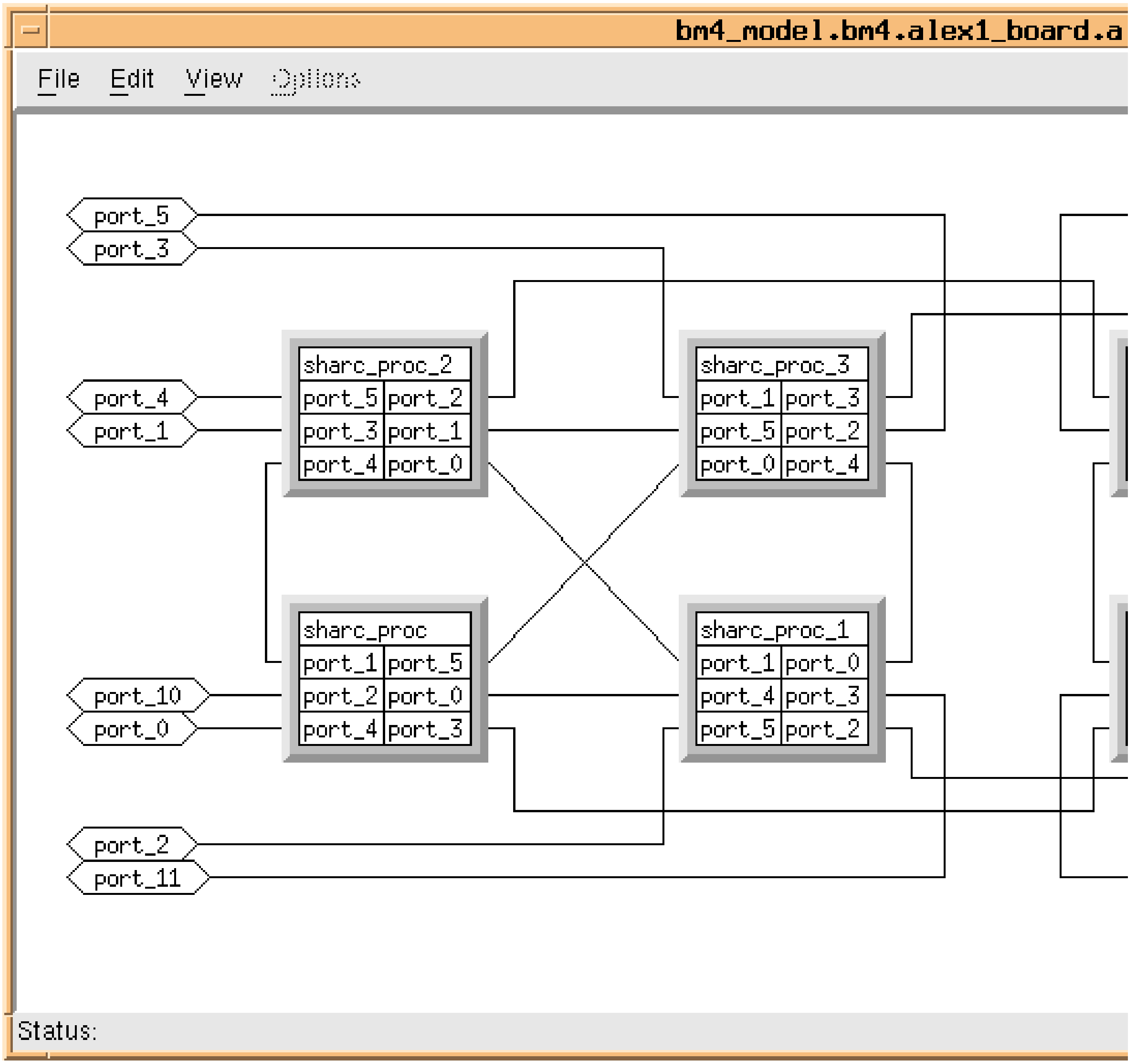
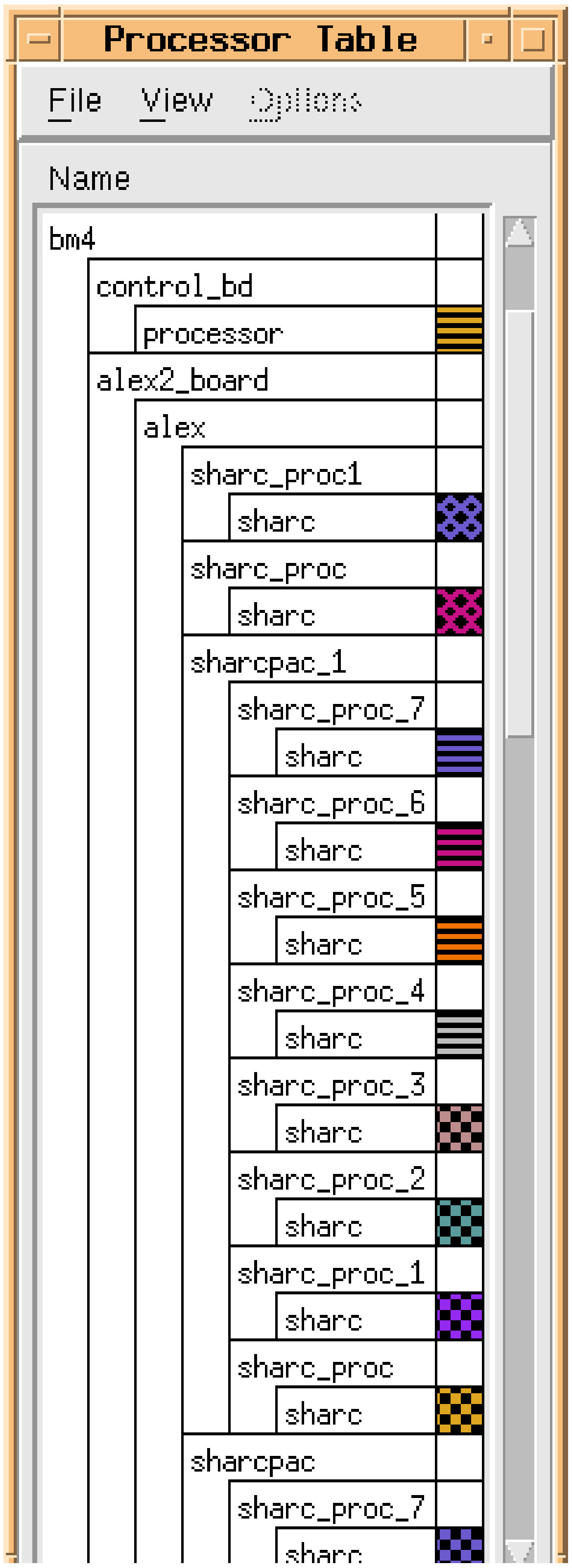


Figure 4 - 4: Graphical model of eight processor SHARCpac.

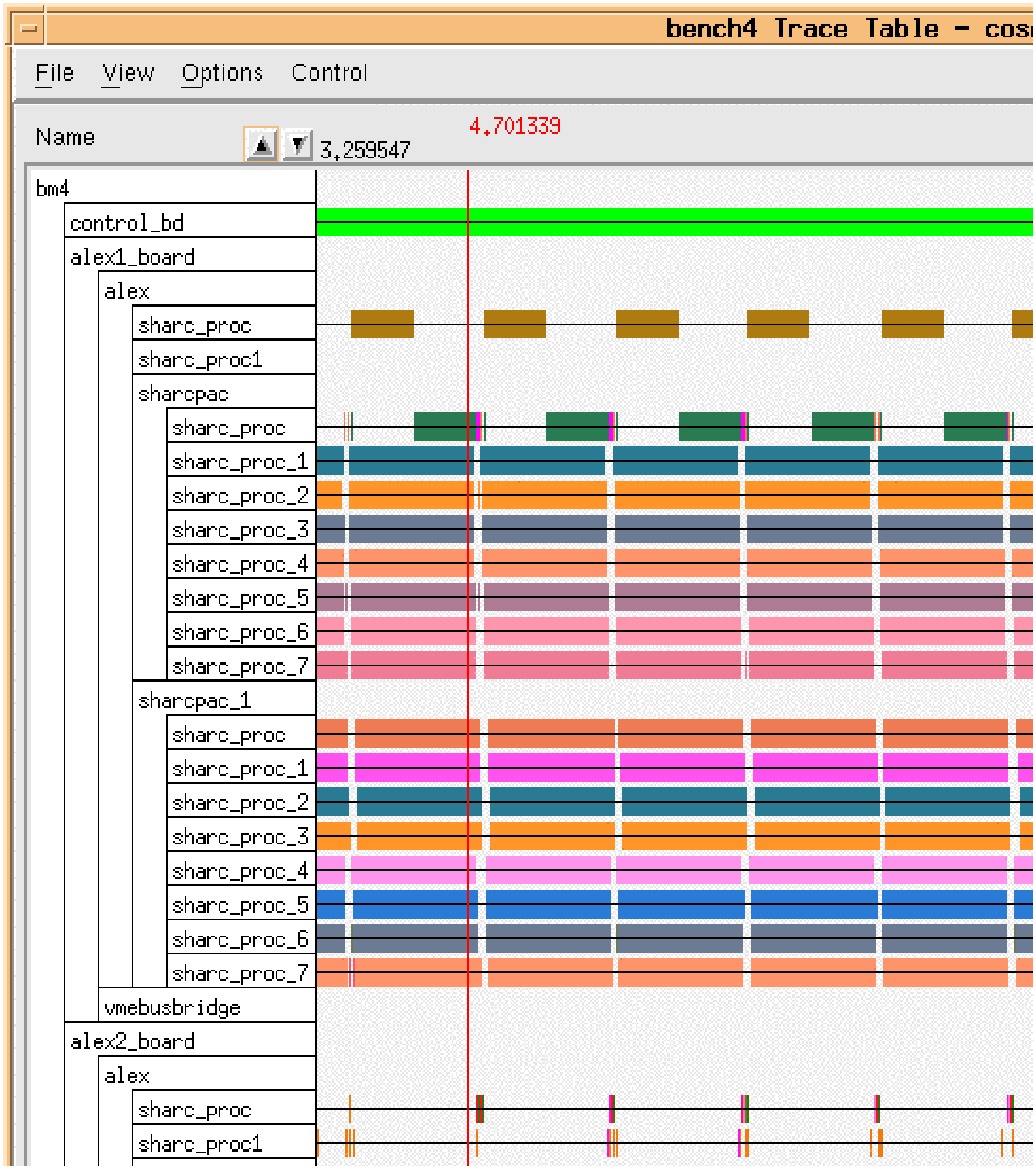
4.3 Architecture tradeoffs

The application graph is mapped to the processors in the architecture using the GEDAE™ Processor Table shown in Figure 4 - 5. The Processor Table is automatically constructed from the graphcial description of the hardware architecture. The elements from which the architecture is constructed are the hardware models supported by the Cosmos tool which have been loaded into the GEDAE™ architecure element library. The nodes of the GEDAE™ software graph are mapped to the individual processors in the processor table using a drag and drop operation. When the mapping is complete, the software graph is translated to pseudocode which provides the stimulus for the Cosmos performance simulation. The hardware architecture is translated to a structural VHDL description, also for use by Cosmos. A control signal is passed to the Cosmos tool which then imports the hardware and software descriptions, generates and executes the VHDL simulation, collects performance event data, and translates the simulation data to the GEDAE™ Trace Table format. When complete, a control signal is passed to GEDAE™ indicating that the results are available. The resulting Trace Table is shown in Figure 4 - 6. The software, architecture, and mapping of the software to the architecture can all be iteratively optimized to achieve an acceptable solution.



```
bm4
  control_bd
    processor          ▤
  alex2_board
    alex
      sharc_proc1
        sharc          ▩
      sharc_proc
        sharc          ▩
      sharcpac_1
        sharc_proc_7
          sharc        ▤
        sharc_proc_6
          sharc        ▤
        sharc_proc_5
          sharc        ▤
        sharc_proc_4
          sharc        ▤
        sharc_proc_3
          sharc        ▦
        sharc_proc_2
          sharc        ▦
        sharc_proc_1
          sharc        ▦
        sharc_proc
          sharc        ▦
      sharcpac
        sharc_proc_7
          sharc        ▦
```

Figure 4 - 5: GEDAE™ Processor Table used when mapping software graph to the graphical hardware description



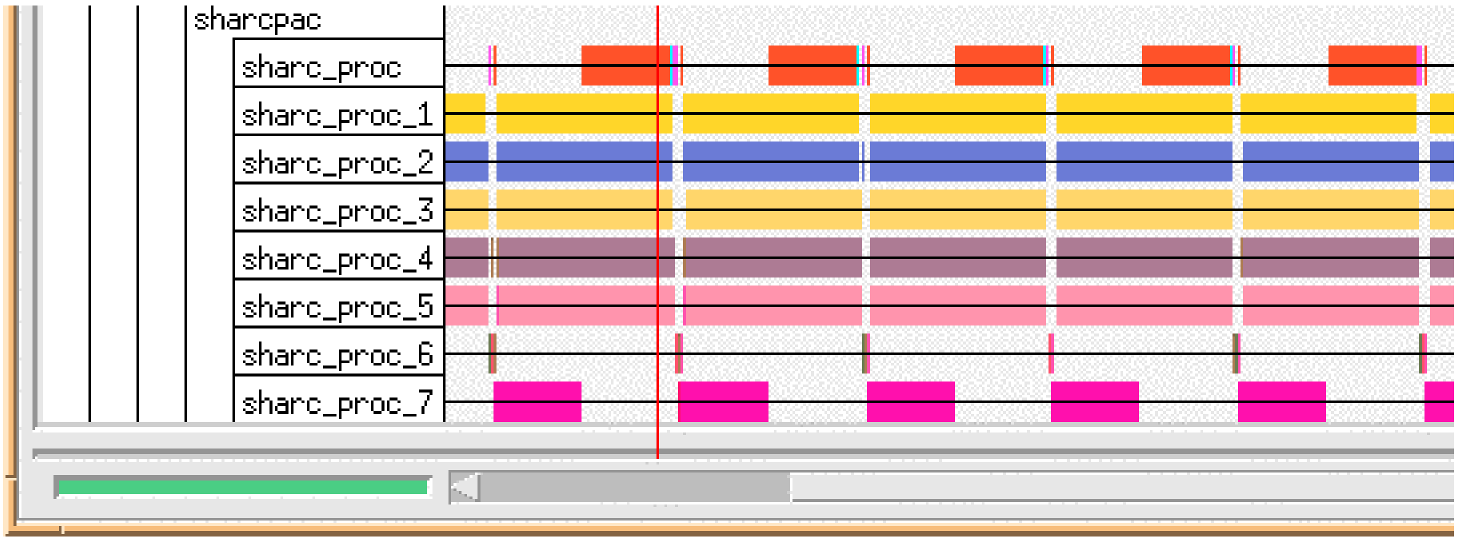


Figure 4 - 6: GEDAE™ Trace Table (partial) representing the simulated performance of the SAIP processing on SHARC architecture from Alex Computer Systems.

4.4 Control software development

The major issue in command program source code development for the SAIP application was the development of the UI layer. Three options were considered:

1. Use Verilog's ObjectGEODE graphical programming and simulation tool
2. Reuse MIT Lincoln Laboratory (MIT LL) Executable Specification Software
3. Use traditional handcoding

The evaluation of these options began with a careful review of the executable specification (several C++ programs) provided by MIT/LL, since it contained the detailed requirements to be levied on the CP. The review concluded that reuse of a portion of the executable specification was the most technically sound, least risk and most cost effective approach. The executable specification was found to be well structured and contained a program that already performed all of the UI level functions. Significantly, this program was already integrated with the MIT/LL Test Bench Software. Due to the clean structure of the MIT/LL software it was determined that AIB generated ASI calls could be easily incorporated and that the modification required to interact with the graphs could be included without altering the Test Bench interfaces or the underlying Inter Processor Communication (IPC) software. The use of ObjectGEODE was rejected because the strengths of the tool were not well matched to this particular CP problem. One of the strengths of ObjectGEODE is managing communications among concurrent threads. In this case the concurrent threads would naturally be the command program, and the Test Bench processes. To use ObjectGEODE on only the CP program portion of the application would forfeit a major strength of the tool. A second ObjectGEODE forte is representing heavily state oriented applications, but in this case the EFSM layer was trivial so again the application was not well matched to ObjectGEODE. Since the MIT/LL provided software was a cleanly written, easily understood small program with a clean interface into the IPC software it was clear that there would be significantly lower labor costs and risks to modify this software than to rewrite it from scratch.

For details of the command program development on the Benchmark 4 effort, see the Benchmark 4 SAIP Case Study.

Next Up Previous Contents

Next: 5 Hardware / Software Codesign Process Applied to Mixed COTS / Custom Architecture **Up:** Appnotes Index **Previous:**3 RASSP Hardware / Software Codesign Process

RASSP Hardware / Software Codesign Application Note

5.0 Hardware / Software Codesign Process Applied to Mixed COTS / Custom Architecture

5.1 SAR Benchmark Application

The RASSP SAR algorithm provides high- resolution, all weather images that can be used to identify main-made objects that are on the ground or in the air. Such object identification typically requires SAR processing to be performed in real- time by means of an embedded signal processor. This application is based on the MIT Lincoln Laboratory Advanced Detection Technology Sensor (ADTS), which is a Ka band SAR (33.56 GHz) sensor. The ADST system consists of an integrated radar, navigation, and recording system. The form factor for the RASSP SAR processor is consistent with operation on board the Amber Unmanned Autonomous Vehicle (UAV).

The RASSP benchmark SAR generates a strip SAR map with a squint angle 90 degrees to the platform velocity vector. It covers a swath of 375 meters at a range of 7.26 K meters. The ADST radar is a fully polarimetric air- to- ground SAR. The radar transmit polarization alternates between horizontal polarity (H-pol) and vertical polarity (V- pol), while H- pol and V- pol are simultaneously processed in the receiver. The radar transmits at a 3kHz rate, so that the same- polarization pulse repetition frequency (PRF) is 1.5 kHz. The SAR resolution is 0.3 meters. The algorithm processes three transmit- receive polarization pairs: HH, VV, and either HV or VH.

For the RASSP benchmark, Lincoln Laboratories provided both an executable specification, written in VHDL, and a list of key system requirements. The system requirements for the RASSP SAR benchmark followed from its intended use to form images of the Earth surface from an airborne platform (Amber UAV) and use them for ATR. These requirements include:

Pulse Repetition Frequency: The physics of the radar transceiver and the system environment may determine the PRF (e.g. the distance to the target defines a minimum PRF).

Acceptable error from "gold" functional output: The RASSP benchmark executable specification developed by Lincoln Labs was delivered with test input data and the corresponding "gold" functional output. This information is essential for testing finite word length effects.

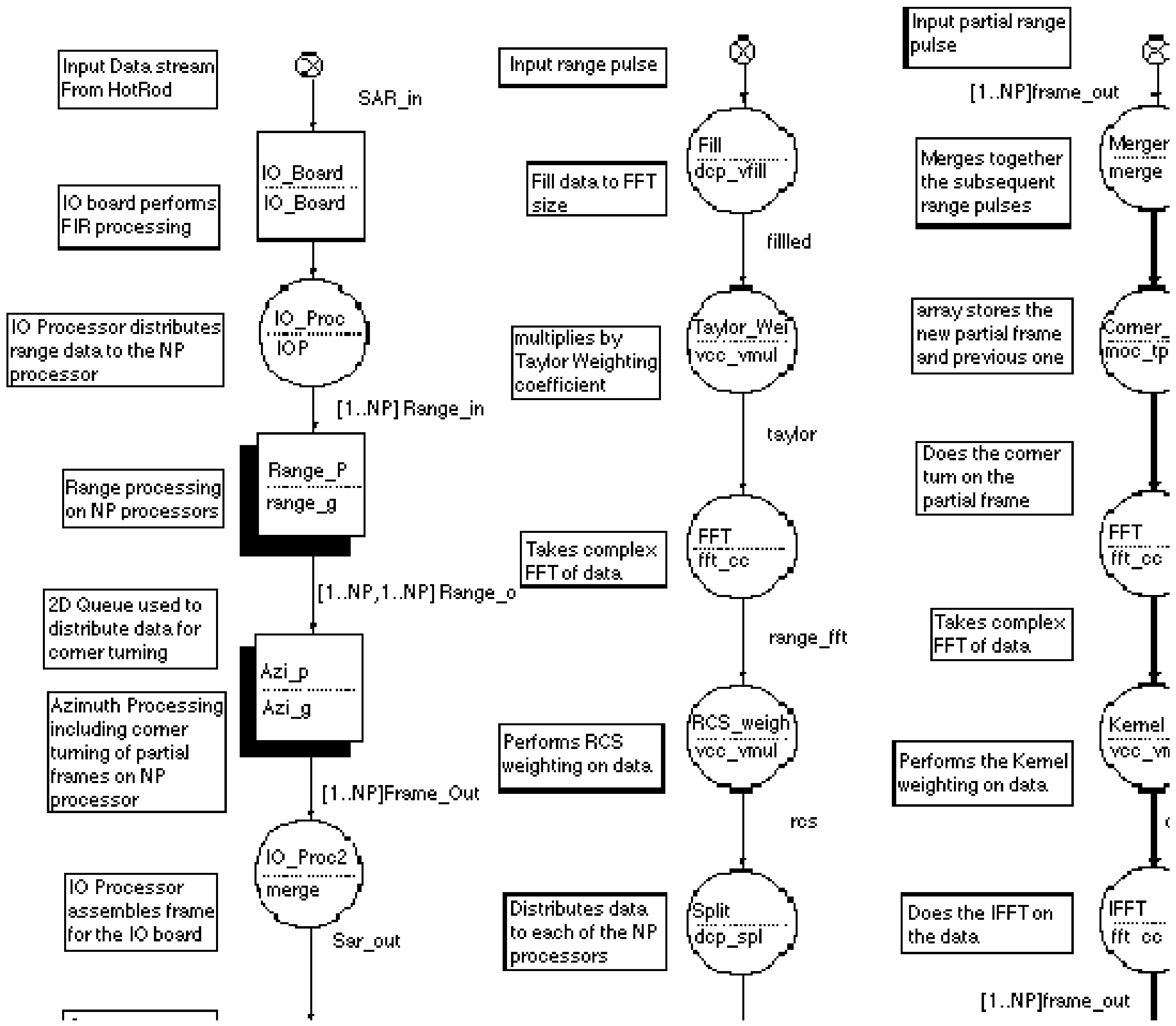
Acceptable latency: Latency for this algorithm is defined as the time from the input of the last pulse of a frame to the display of last row of the output image.

Acceptable utilization of hardware components: A system specification often defines some growth margins in terms of component utilization. Growth margins are an important factor in RASSP designs, since model- year upgrades can cause growth in processing loads.

Acceptable levels of memory occupancy: A system specification often defines some growth margins in terms of available memory. This is another growth margin that is important in RASSP systems where model- year upgrades are anticipated.

5.2 Application of Hardware / Software Codesign

The SAR application in the Processing Graph Method (PGM) notation is shown in Figure 5 - 1a. The radar data stream is received by the IO_Board in a series of 512 sequential range pulses of 2032 complex numbers. Each range pulse is FIR filtered by the IO Board and then a separate IO process distributes each subsequent pulse to NP processes. This is repeated for all of the 512 pulses in each image frame. The range processing which is defined in the Range subgraph in Figure 5 - 1b proceeds for each of the NP range. At the end of the range processing the range pulse is divided into NP segments and each of the NP - 1 segments are sent to the appropriate azimuth processes. The PGM mechanism that represents this data distribution process is a two dimensional family queue indicated on the figure as [1..NP,1..NP]Range_o. Families are noted in PGM by the shaded nodes. The data from the range processing is loaded into one of the dimensions of the queue associated with that range family. There are NP of these one dimensional queues for a total of 2 * NP queues which are represented by the two dimensional family shown in Figure 5 - 1a. The azimuth process first merges the data from the NP queues. The data is then corner turned and the azimuth processing shown in Figure 5 - 1c is completed on this partial data set. After each partial azimuth frame is completed the IO process (IO_Proc2) moves these partial frames sequentially back to the IO board.



Input Data stream From HotRod
SAR_in
IO_Board
IO_Board
IO board performs FIR processing
IO_Proc
IOP
IO Processor distributes range data to the NP processor
[1..NP] Range_in
Range_P
range_g
Range processing on NP processors
[1..NP,1..NP] Range_o
2D Queue used to distribute data for corner turning
Azi_p
Azi_g
Azimuth Processing including corner turning of partial frames on NP processor
[1..NP]Frame_Out
IO_Proc2
merge
IO Processor assembles frame for the IO board
Sar_out
Input range pulse
Fill
dcp_vfill
Fill data to FFT size
filled
Taylor_Wei
vcc_vmul
multiplies by Taylor Weighting coefficient
taylor
FFT
fft_cc
Takes complex FFT of data
range_fft
RCS_weigh
vcc_vmul
Performs RCS weighting on data
rcs
Split
dcp_spl
Distributes data to each of the NP processors
Input partial range pulse
[1..NP]frame_out
Merger
merge
Merges together the subsequent range pulses
Corner_
moc_tp
array stores the new partial frame and previous one
Does the corner turn on the partial frame
FFT
fft_cc
Takes complex FFT of data
Kernel
vcc_vn
Performs the Kernel weighting on data
IFFT
fft_cc
Does the IFFT on the data
[1..NP]frame_out

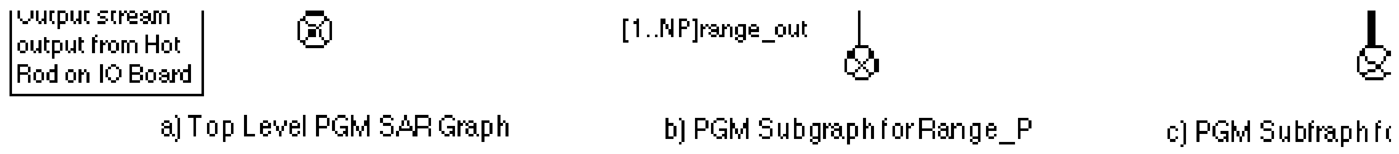

Output stream output from Hot Rod on IO Board
[1..NP]range_out
a) Top Level PGM SAR Graph
b) PGM Subgraph for Range_P
c) PGM Subfraph fc

Figure 5 - 1: PGM Model of Single Polarity SAR Algorithm

A key element of the LM- ATL RASSP design process which spans both the architecture process and the detailed design process is virtual prototyping. The various levels of the virtual prototype of the SAR system were used in a hierarchical fashion to quickly assess design risks involved in integrating the hardware and software elements and minimize or eliminate those risks. The first level of virtual prototype was a network performance model which used VHDL to describe data communications at the packet level transfer and modeled the application code with a pseudo- code which was interpreted by the computational element of the processor model. This allowed true hardware/software codesign with the independent specification of the software and hardware. Each of the candidate architectures was evaluated to ensure minimum system sizing (number of processors and amount of memory), and optimum software mapping, and to eliminate performance bottlenecks of the interprocessor communication network.

The second level of virtual prototype added functionality to the performance (timing and structure) models. A data field added to the network token allowed actual data passing between the models, and the high level pseudo- code modeling of the software running on the processors used in the performance simulation was replaced with processes making DSP math library calls.

The third level of virtual prototype described the custom modules down to individual components at the behavioral level with emphasis on interface behavior rather than internal chip structure, and described the FPGAs in synthesizeable VHDL at the RTL level. (Details of the performance modeling and virtual prototyping are found in the "Token- Based Performance Modeling" and "Virtual Prototyping Concepts" Application Notes.)

Processor behavioral and performance simulations using the VHDL virtual prototype support trade- offs. Mixed levels of simulation (algorithm, abstract behavioral, performance, ISA, RTL, etc.) are used to verify interaction of the hardware and software. These models are composed largely of hierarchical VHDL models of the architecture. We choose the models, to the maximum extent possible, from the MYA elements in the RASSP reuse library. The RASSP team develops and inserts new required library elements into the reuse library to support this design phase. The executable specification has now evolved into a more detailed set of functional and performance models that are architecture- specific. Software algorithm implementations are also now specific to the candidate architecture(s).

Lockheed Martin ATL evaluated two COTS- based hardware architectures for the RASSP SAR benchmark system both based on Mercury Raceway configurations. Mercury provides several processor options for its MCV6 system, including two considered in these Hardware/Software configuration tradeoffs: the Intel i860 processor, and the Analog Devices ADSP21062 SHARC processor. The two configurations considered were:

- Five MCV6 boards, each with two daughter cards holding 2 I860 processors (a total of 20 I860 processors), plus a custom board (Data I/O module) doing the FIR filtering and unpacking.

- Three MCV6 boards, one with a 2 i860 daughter card and a daughter card with 4 SHARC's, two with one daughter card each having 4 SHARCs (a total of 12 - 21060 processors and one i860 needed for control), plus a custom board doing the FIR filtering and unpacking.

Example results of token- based performance modeling and simulation performed on a twelve (12) processor SHARC architecture are shown in Figures 5 - 2 and 5 - 3. Figure 5 - 2 shows a high level graphical depiction of the software corresponding to three polarizations of SAR processing and a graphical depiction of the hardware being simulated. The hardware view shows the top level architecture consisting of an I/O board and three Sharc boards. Each Sharc board is hierarchical and can be expanded as shown in the figure. The software view represents the processing in the form of a source of data, twelve identical range processing functions, twelve identical azimuth processing functions, and a data sink. The mapping of the algorithm is such that a range processing function and the corresponding azimuth processing function are mapped to one of the twelve SHARC processors in the architecture. The interconnections between four range functions and four azimuth functions represents a distributed corner turn which is necessary due to the size of the images being processed and the limited memory associated with each processor. The three interconnected groups of range and azimuth functions represents identical processing for the three polarizations required.

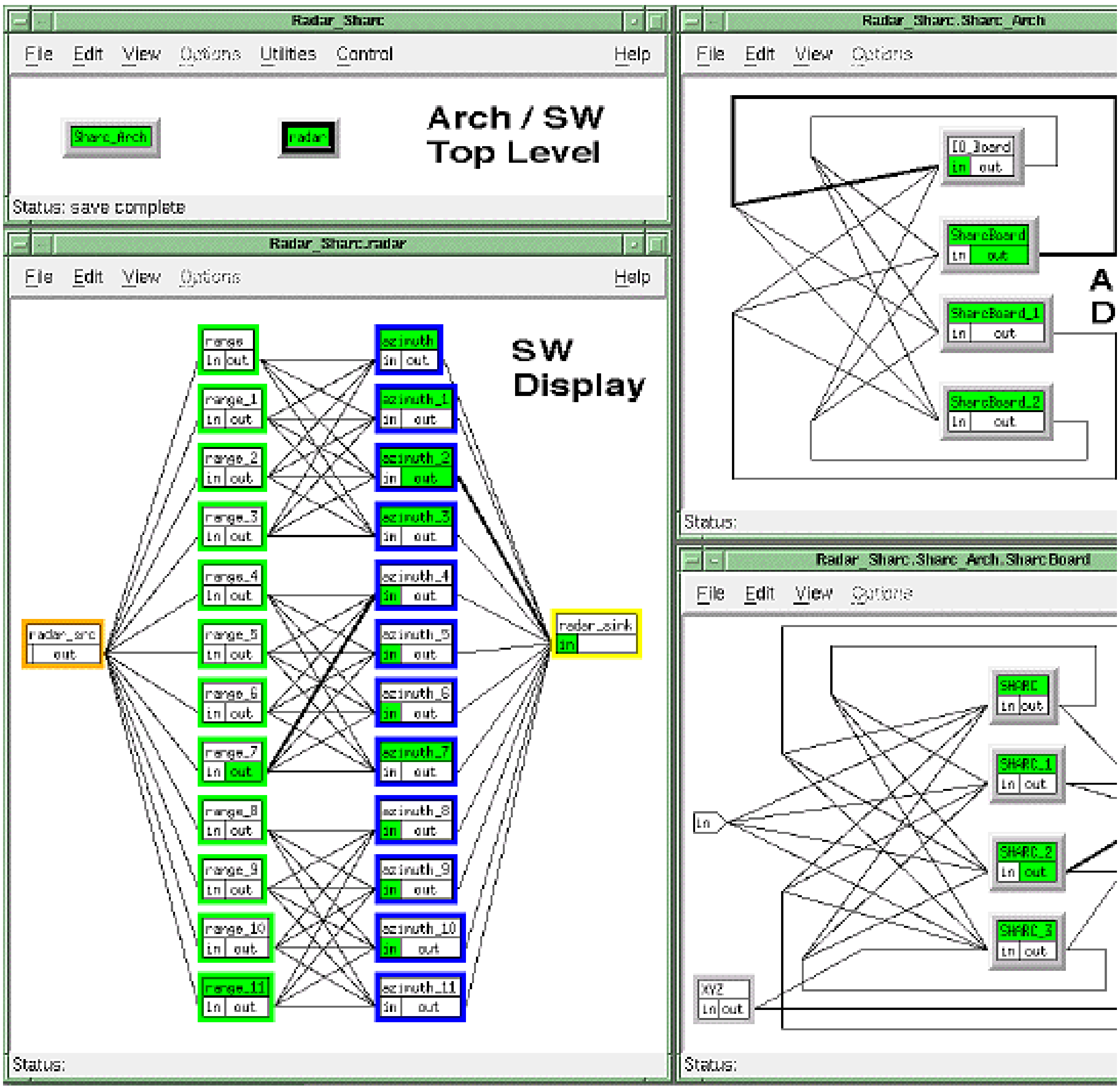
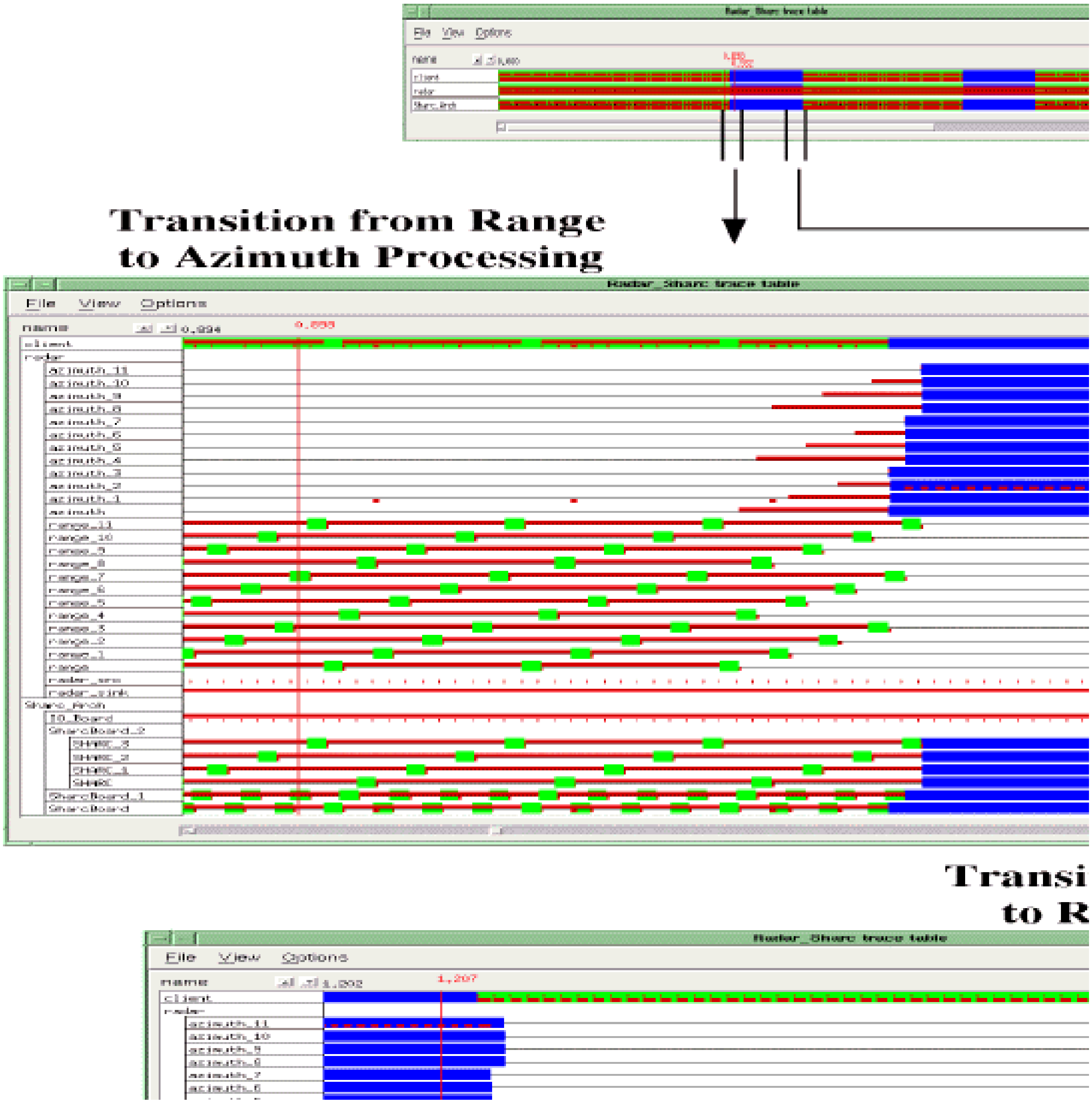


Figure 5 - 2: Graphical view of software for three polarization SAR application and one SHARC hardware architecture

Figure 5 - 3 shows the top level view of the simulation results in the form of a timeline, along with two

expanded views corresponding to interesting transition areas. In each expanded timeline there is a line corresponding to each of the twelve range and azimuth functions in the software graph as well as the source and sink functions. There is also one line for the I/O board and the three SHARC boards, with one of the SHARC boards expanded to show the four individual SHARC processors. The images in Figures 5 - 2 and 5 - 3 are created using the Architecture Definition and Visualization Tool (ADVT) which also provides the ability to "replay" the simulation on the graphical representations of the hardware and software for analysis purposes. The simulation was performed using VHDL and the simulator output postprocessed for insertion into ADVT. The highlighted inputs or outputs on either the hardware or software graph in Figure 5 - 2 indicates that a data transfer is in process or pending. A highlighted title bar indicates that the particular software process is active or the architecture resource is busy. A highlighted connection indicates that data communication is in progress.



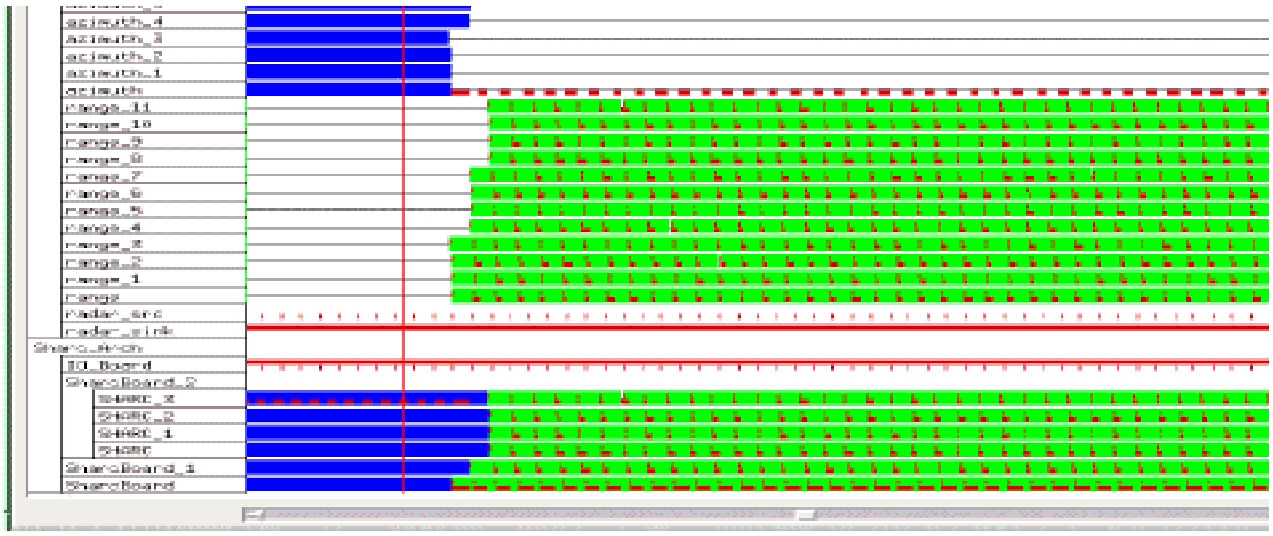


Figure 5 - 3: VHDL simulation results displayed in performance timelines

The LM- ATL selected SAR signal processor architecture, shown in Figure 5 - 4, contains four major architectural elements:

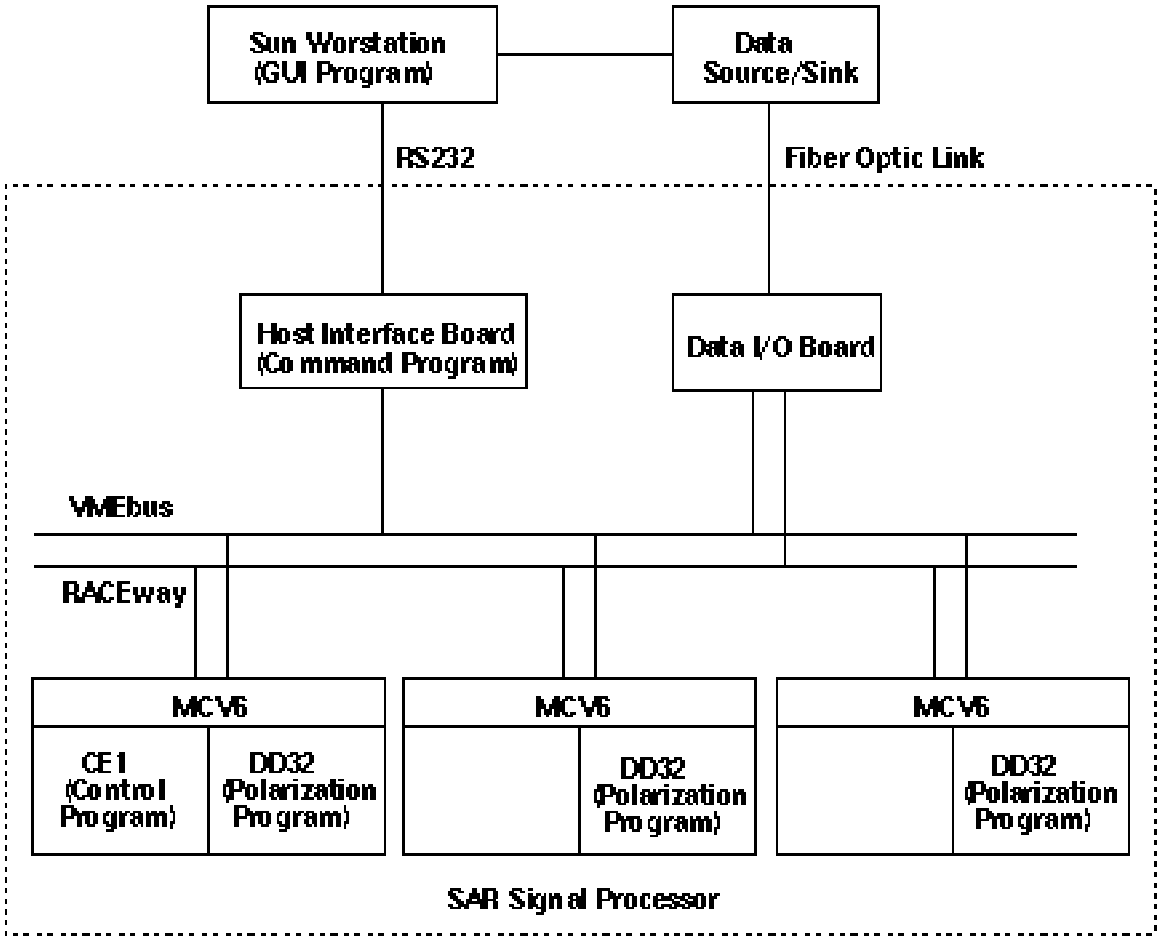

Sun Worstation (GUI Program)
Data Source/Sink
RS232
Fiber Optic Link
Host Interface Board (Command Program)
Data I/O Board
VMEbus
RACEway
MCV6
CE1 (Control Program)
DD32 (Polarization Program)
MCV6
DD32 (Polarization Program)
MCV6
DD32 (Polarization Program)
SAR Signal Processor

Figure 5 - 4: Lockheed Martin ATL SHARC SAR Processor Architecture

Mercury Computer Systems MCV6 Processor Boards (VME 6U format) perform the bulk of the signal processing. Each MCV6 can have up to two daughtercards. The SAR signal processor design uses a single daughtercard containing four Analog Devices ADSP21062 SHARC processor chips with 32 MBytes of DRAM for the processing of each polarization.

A 68040 based single board computer (SBC), the Motorola MVME162, serves as the host interface and controls the SAR signal processor operation. When the SAR signal processor is in stand- alone mode, the 68040 boots the SAR signal processor and controls its operation.

A custom designed Data I/O Board interfaces the SAR signal processor to the radar data source and sink, and performs front end signal processing functions. This includes synchronizing operation based upon the preamble sent with each pulse, performing video- to- baseband I/Q conversion, FIR filtering, and keeping track of pulse, polarization, and frame boundaries.

An interconnect network consisting of Mercury Computer Systems Raceway acts as a high- bandwidth, point-to- point network for data transfer, and the VMEbus performs control operations.

The main risk with the chosen architecture was that the ADSP21062 SHARC processors were not yet available when the choice was made. Therefore, a risk mitigation plan was established to use dual i860 daughtercards for the signal processing if the SHARC based daughtercards were not available at the time of system integration. A single polarization design requires two i860 boards, each with two daughtercards, and a three polarization design requires five i860 boards, each with two daughtercards. Because of SHARC processor unavailability in the time remaining for iimplementation prior to the Second Annual RASSP Conference, a single polarization, I860 based architecture was developed.

Software development in the architecture process deviates significantly from traditional approaches. The functionality which has been allocated to software can be broken into three major areas:

1. algorithm, as specified in the data flow graph;
2. scheduling, communications, and execution, as specified by the mapping of the graph to a specific architecture; and
3. general command/control and support software. The RASSP program is automating the first two to the maximum extent possible and investigating the potential for automation of the third.

This is accomplished using a graph- based programming approach(es) that supports correct- by- construction software development based on algorithm and architecture- specific support library elements.

Mangement Communications and Control Inc (MCCI), a Lockheed Martin ATL RASSP subcontractor has developed an autocoding system which generates the software automatically from the PGM data flow graph description. The autocoding process is driven from:

- the data flow graph,
- description of the architecture, and
- the mapping of the processing graph to the architecture.

The autocoded graph partitions are executed under the control of a Static Run- Time System which provides graph management facilities, a reusable interprocessor communication substrate, and a well defined external interface for control.

The third major process is the detailed design of software and hardware elements. As with the system and architecture processes, detailed design is carried out and verified for both hardware and software via a set of detailed functional and performance simulations. At the completion of this process, the design is established.

The RASSP software development methodology was used to develop the SAR benchmark, but since the RASSP autocoding tools were not yet available, the SAR application software had to be hand coded. Subsequent to completion of the hand coding, the MCCI autocode tool was completed and delivered to ATL. In order to exercise the tool, it was used to reimplement the application software portion of the SAR problem. The results of this exercise were very impressive.The Autocode Tools truly output a complete solution.

Loading and starting the processes was performed flawlessly by the tool. In addition, all memory management and interprocessor communications is included in the generated code. It is important to note that all interprocessor communication was provided by the Run- Time System and that the application developer had to write no communication software to achieve correct interprocessor communication the first time. This exercise was the first time the RASSP software methodology was demonstrated. Overall, the exercise was a success and marked a major milestone for the RASSP program.

The exercise lasted approximately one month. After generating the required PGM graphs, the tools automatically generated all code necessary to implement the graphs on the target hardware. The only thing which needed to be hand coded was the IO Procedure which interfaced the application graph to the ATL designed IO Board. The tool provides a set of library services to support the IO Procedure to application graph interface. System integration took approximately 2 weeks. This is significantly less time than was required to integrate and test the hand coded software. The autocoded software's execution efficiency was within 15% of the hand coded software. Although memory efficiency was not as good, a number of areas were identified and subsequently improved.

During the hardware portion of the detailed design process, behavioral specifications of the processor are transformed into detailed designs (RTL and/or logic- level) through a combination of hardware partitioning, parts selection, and synthesis. Detailed designs are functionally verified using integrated simulators, and performance/timing is also verified to ensure proper performance. The process results in detailed hardware layouts and artwork, net lists, and test vectors which can then be seamlessly transitioned to manufacturing and test via format conversion of the data.

The hardware design for the SAR signal processor consisted of the mechanical design of the chassis and the design of the data I/O Board. All other boards in the SAR signal processor are off the shelf modules which required no new hardware design. Hence, the hardware design discussion focuses on design of the Data I/O Board shown in block diagram form in Figure 5 - 5.

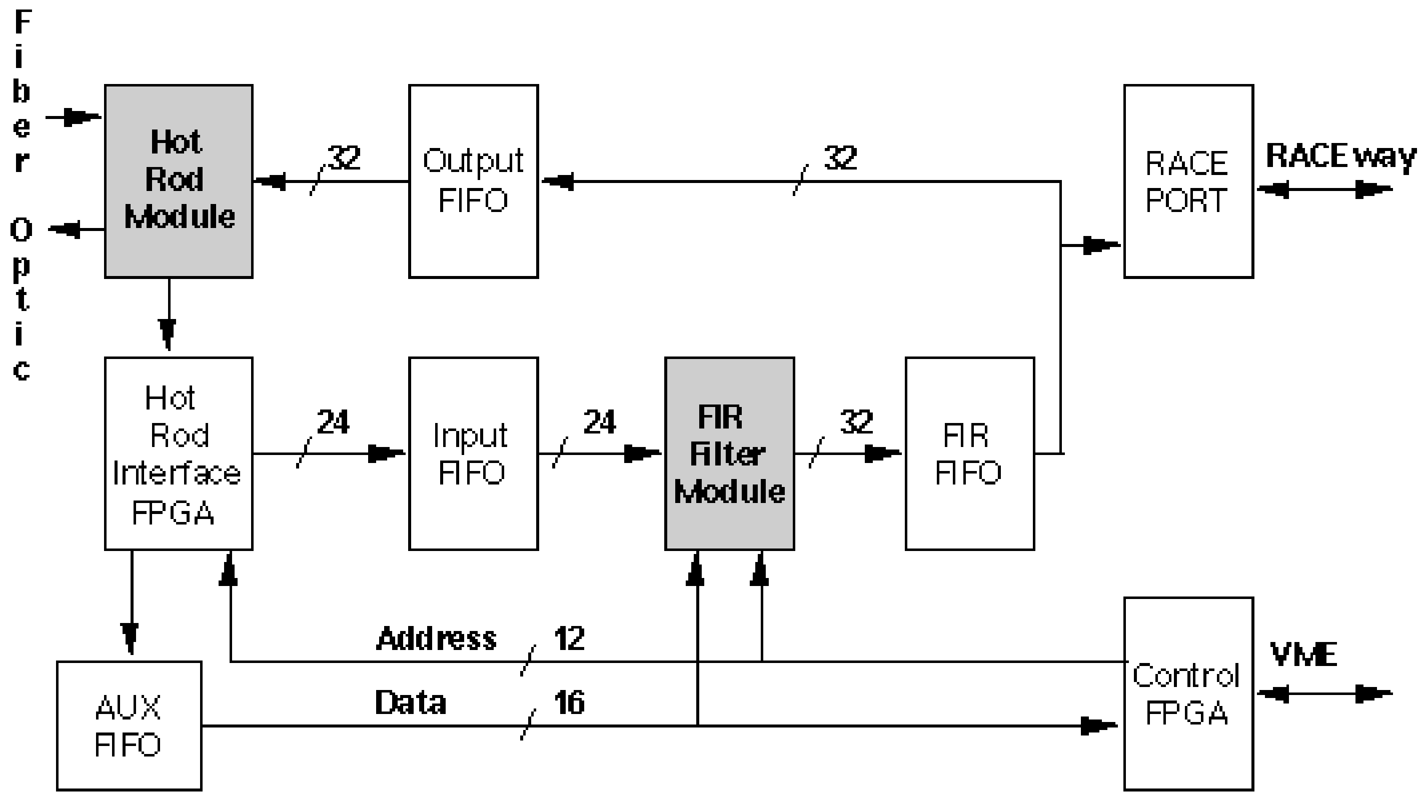


Figure 5 - 5: Data I/O board block diagram

The Hot Rod module (HRC- 500FS Fiber Optic Interface Card) provides separate fiber optic interfaces for the input radar data and output image data. The Hot Rod daughtercard interfaces to the rest of the Data I/O Board through two 40- bit wide buses - one for transmit and one for receive. The input data rate of 4.56 Megawords and output data rate of 6.83 Megawords are both within the 11.25 Megawords maximum of the Hot Rod. The Hot Rod has an internal loopback mode that connects the transmit side back to the receive side. The loopback mode is used during Data I/O Board

The Hot Rod Interface FPGA performs a number of operations on the radar data in addition to controlling the Hot Rod. The Hot Rod Interface FPGA contains logic that looks for the preamble that defines the beginning of a radar pulse. Once the start of a pulse is detected, the odd and even data samples are extracted along with bit serial data defining polarization and auxiliary radar data. The 12- bit odd and even pulse samples are modulated by (- 1)n before being written into the Input FIFO. The auxiliary radar data is written to the AUX FIFO. The number of words, pulses, and image frames in both the receive and transmit channels are counted as part of the I/O control.

The FIR filter daughtercard is capable of simultaneously processing I and Q channel data at a 5 MHz input rate. The input data is 12- bit twos complement, the filter coefficients are 23- bit twos complement, and the output is 32- bit twos complement in both I and Q. Each channel uses two Plessey PDSP16256 Programmable FIR Filter chips configurable with up to 64 taps, with one FIR chip processing the most significant portion of the filter coefficients, and the other processing the less significant portion of the filter coefficients. The 16K deep FIR FIFO buffers received data until it is sent out of the RACE interface.

The RACE port interface contains two ASICs designed by Mercury Computer Systems, Inc. The RACE interface sends the received radar data to the appropriate signal processor, and receives output image data from the signal processors. The 4k- deep output FIFO provides data buffering between the RACE interface and the Hot Rod transmit port.

The Control FPGA controls the operation of the FIR filter and the FIFOs, keeps track of the status of all FIFOs, and implements the VMEbus interface.

Test considerations played an important part in the design of the Data I/O Board and associated FPGAs. One testability issue is the lack of JTAG scan on a majority of the COTS components. The design approach taken was to add JTAG scan bus transceivers to signal paths between non- JTAG devices. A second testability issue is the presence of asynchronous interfaces. The approach taken is to design these interfaces so that they can be tested synchronously, and to add test modes that force synchronous operation. Test modes included in the FPGA design enable bypass of either the PRI detection logic or the FIR Filter. FIR bypass is significant in that it allows testing of the Data I/O motherboard without the FIR Filter daughterboard in place. FPGA test modes reduce the cycle time of high modulo counters, allowing testing of higher order counter bits. The various test modes, which are part of the VHDL model, are included in the data I/O functional simulations.

Synopsys was used to synthesize logic for the two FPGAs using the AT&T ORCA cell library. NeoCAD was then used to map these cells to the ATT2C15 FPGA programmable logic cells (PLCs), place the cells, and route the interconnect between the cells. Static timing analysis (using NeoCAD) after mapping, placement, and routing was performed to identify any nets not meeting timing and speed specifications. In some cases, it was necessary to go back and modify the VHDL description to meet timing and speed specifications. A typical modification was the introduction of pipelining on paths not meeting speed. Once the FPGA design was completed, the logic and timing was back- annotated into the board level VHDL model and functionality was reverified.

Detailed schematics of the Data I/O Board design were captured using Mentor Graphics Design Architect. The schematics could not be finished until the FPGA design was completed with all pin assignments finalized. Once the schematics were finalized, placement and routing using Mentor Boardstation were performed for the Data I/O motherboard and FIR filter daughterboard. The board layout parasitics were back- annotated into the VHDL model to confirm that functional and timing specifications are met.

Since most of the software developments are verified during the architecture phase, they are limited at this point to generation of those elements that are target- specific. This includes configuration files, bootstrap and download code, target- specific test code, etc. All the software is compiled and verified (to the extent possible) on the final virtual prototype prior to the detailed design review. Design release to manufacturing marks the end of the RASSP design process.

Next Up Previous Contents

Next: 6 Lessons Learned **Up:** Appnotes Index **Previous:**4 Integrated Toolset Example for COTS Architecture

Approved for Public Release; Distribution Unlimited *Dennis Basara*

RASSP Hardware / Software Codesign Application Note

6.0 Lessons Learned

Some of the lessons learned during the RASSP benchmarking process about Hardware/Software Codesign are described. As stated at the outset, Hardware/Software Codesign is the co-development and co-verification of hardware and software through the use of simulation and/or emulation. Based upon the RASSP Benchmarks, the following observations are made:

6.1 Model validation

Model validation is a crucial step with any simulation or analysis activity. Validation occurs in two ways, component validation and process validation. Component validation is the verification of individual models and their characterizations. Process validation is verifying the results of a system model against actual numbers from the resulting system. Individual leaf cell (lowest level model) verification is relatively straightforward. Leaf cells typically are characterized to model different types of components such as memories, ALUs, etc. These can be verified through simple testbenches and for off the shelf parts, timing numbers are readily available. As the design process progresses, a performance model's accuracy should be continually checked against more detailed models as they become available or measurements from the actual components. Any mismatch should be corrected to maintain the performance model's accuracy, to test for continued compliance with requirements, and to support subsequent re-use and model-year upgrades. This activity departs from traditional processes which do not maintain -and therefore effectively discard- the performance model once the architecture design has completed.

6.2 Simulations must be rapid

When conducting performance simulations early in the design process, the simulations must be rapid. However, rapidly simulating a significant portion of the real-time SAR application executing on the full system composed of as many as 24 PE's and crossbar elements required a much higher efficiency and modeling abstraction than that of the typical ISA-level model. To be abstract, yet accurate, only the necessary details were resolved in the model. These included significant protocol events such as initiation and termination of data transfers as well as significant computational events such as the begin and end of bounded computational tasks. The resolved events focus around contention for computation and communication resources whose usage time, once allocated for a task or transfer, was highly deterministic. The simulation is valuable because the contention for the multiple resources is not conveniently predictable.

6.3 Tools must maximize useful information to the designer

The design group produced time-line graphs from the simulation results which showed the history of task executions on the PE's. The graphs were useful in helping to visualize and understand the impact of mapping options that led the design group to modify, optimize, and ultimately verify the partitioning, allocation, and scheduling of the software tasks onto the hardware elements. The time-line graphs showed the times when PE's were idle due to data starvation or buffer saturation that helped isolate other resource contentions and bottle-necks. Plots of memory allocation as a function of time were also valuable in visualizing and balancing the extent of memory usage throughout the algorithm execution. The resultant software task partitioning and schedules led directly to the production of the target source code through a straight-forward translation into sub-routine calls. Because the ultimate implementation became the sum of time-predictable events for which linear additivity basically holds, it was not surprising that the simulation results accurately predicted the physical system's actual run-time performance within a few percent.

6.4 New tools and tool interoperability required

A major portion of the ATL RASSP program was directed toward the development of new tools as well as providing interoperability and seamless transition between tools. The execution of the benchmark program convinced us that the overall approach taken on the RASSP program is valid. New tools at the architecture definition level which facilitate the rapid tradeoff of candidate architectures by predicting performance of the application software on the virtual architecture improve the early design process. The direct coupling of these high level tools with autocoding tools that provide the ability to generate target software automatically which is comparable in performance to hand generated software greatly improves the software development process.

6.5 Software generation for multiprocessor systems will get easier

Software for signal processing including the interprocessor communications on large systems can and will be automatically generated with nearly the same efficiency as hand coding for many applications. Experiments on the RASSP Benchmarks using these new tools indicate that such tools are capable of eliminating the software development associated with interprocessor communication which is where the majority of integration and test time is expended. This elimination of interprocessor communication software development, coupled with graph based application development will provide an order of magnitude productivity improvement in signal processing software generation. Furthermore the productivity improvement ssociated with retargeting an application from one architecture to another will be even greater.

6.6 Software development paradigm shift is required

In order to achieve the very large productivity improvements promised by the utilization of autocoding tools, the software developers must begin to think in a data flow paradigm. If the overall signal processing required is defined in a data flow paradigm using tools designed for this purpose, then architecture tradeoffs naturally flow from this same description; detailed functionality is achieved by systematically incorporating the functionality into the data flow description; and the fully functional data flow graph also drives the automatic code generation process which is supported by a vendor supported run-time kernel that is compatible with the generated autocode. When implemented properly, even larger productivity improvement can be achieved when systems must be modified due to algorithmic upgrades or insertion of new hardware technology. When applications are developed and maintained at the data flow graph level, retargetability becomes (in large part) a responsibility of the autocode tool vendor rather than the application developer.

6.7 Legacy systems

There are few systems which are developed from scratch. More often, existing applications are upgraded with additional functionality, new algorithms or new hardware. The transition of legacy software which is perfectly valid to a data flow paradigm can be a major effort and should not be underestimated. Once the viability of autocoding tools is clearly established, algorithm developers will begin to use the autocode tools for algorithm development which will bypass the need for later code conversion. Legacy systems and their existing software must still be dealt with. The best that current technology offers is that the potentially labor intensive conversion of existing code to a data flow description must be done only once - future software or hardware upgrades become much easier. In contrast, without making this shift to a data flow description which is supported by automatic code generation, existing systems will continue to require major software rewrites each time the hardware is upgraded.

Next Up Previous Contents

Next: 7 Further Information and Other Areas of Interest **Up:** Appnotes Index **Previous:**5 Hardware /

Approved for Public Release; Distribution Unlimited *Dennis Basara*

RASSP Hardware / Software Codesign Application Note

7.0 Further Information and Other Areas of Interest

7.1 Trademarks and Copyrights

GEDAE™ is a trademark of Lockheed Martin Corporation. © 1997 Lockheed Martin Coporation. All rights reserved.

Omniview Cosmos is a trademark of Omniview Design Inc. 1998 Omniview Design Inc.

ObjectGEODE is a trademark of VERILOG SA.

7.2 Links to related areas of interest

Applicable Application Notes

- Model Year Architecture
- Methodology
- Executable Specifications
- Token-Based Performance Modeling
- Virtual Prototyping Concepts
- Data Flow Graph
- Autocoding for DSP Control

Applicable Case Studies

- Synthetic Aperture Radar (SAR)
- ETC4ALFS on COTS Processors
- Semi-Automated IMINT Processing (SAIP)

Specifications, Documents

- RASSP Methodology Version 2.0

Other Web sites

- www.gedae.com
- www.omnivw.com/products/cosmos.html
- www.verilogusa.com/og/og.htm

7.3 Articles / Presentations

Pridmore, J. S. and W. B. Schaming, "RASSP Methodology Overview", Proceedings of the 1st Annual RASSP Conference, Arlington, Va. August 15-18, 1994. [PRIDMORE_94]

Myeres, C., A. Bard, and W. B. Schaming, "Hardware/Software Codesign", RASSP working document, October, 1994.

Bard, A. D. and W. B. Schaming, "Hardware/Software Codesign in the Lockheed Martin Advanced Technology Laboratories RASSP Program", Proceedings of the 2nd Annual RASSP Conference, Arlington, Va. July 24-27, 1995. [BARD_95]

Pridgen, J., R. Jaffe, and W. Kline, " RASSP Technology Insertion into the Synthetic Aperture Radar Image processor Application", Proceedings of the 2nd Annual RASSP Conference, Arlington, Va. July 24-27, 1995. PRIDGEN_95]

C. Robbins, " Autocoding in Lockheed Martin ATL/Camden RASSP Hardware / Software Codesign", Proceedings of the 2nd Annual RASSP Conference, Arlington, Va. July 24-27, 1995. [ROBBINS_95]

Hein, C. and D. Nasoff, " VHDL-Based performance Modeling and Virtual Prototyping", Proceedings of the 2nd Annual RASSP Conference, Arlington, Va. July 24-27, 1995. [HEIN_95]

Next Up Previous Contents
Next: Up: Appnotes Index **Previous:**6 Lessons Learned

Approved for Public Release; Distribution Unlimited *Dennis Basara*