WL-TR-96-1142

CAD TOOLS FOR THE DEVELOPMENT AND REUSE OF MODELS OF SIGNAL PROCESSING SOFTWARE AND HARDWARE

VOLUME 1 – FINAL REPORT

G. A. FRANK
B. E. CLARK

CENTER FOR DIGITAL SYSTEMS ENGINEERING
RESEARCH TRIANGLE INSTITUTE
3040 CORNWALLIS ROAD
RESEARCH TRIANGLE PARK NC 27709

F. G. GRAY
J. R. ARMSTRONG

VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY
BLACKBURG VA 24061

JANUARY 1997

FINAL REPORT FOR PERIOD 01 SEP 93 – 01 SEP 96

Approved for public release; distribution unlimited

INFORMATION DIRECTORATE
AIR FORCE RESEARCH LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7334

NOTICE

USING GOVERNMENT DRAWINGS, SPECIFICATIONS, OR OTHER DATA INCLUDED IN THIS DOCUMENT FOR ANY PURPOSE OTHER THAN GOVERNMENT PROCUREMENT DOES NOT IN ANY WAY OBLIGATE THE US GOVERNMENT. THE FACT THAT THE GOVERNMENT FORMULATED OR SUPPLIED THE DRAWINGS, SPECIFICATIONS, OR OTHER DATA DOES NOT LICENSE THE HOLDER OR ANY OTHER PERSON OR CORPORATION; OR CONVEY ANY RIGHTS OR PERMISSION TO MANUFACTURE, USE, OR SELL ANY PATENTED INVENTION THAT MAY RELATE TO THEM.

THIS REPORT IS RELEASABLE TO THE NATIONAL TECHNICAL INFORMATION SERVICE (NTIS). AT NTIS, IT WILL BE AVAILABLE TO THE GENERAL PUBLIC, INCLUDING FOREIGN NATIONS.

THIS TECHNICAL REPORT HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION.

Luis M. Concha

LUIS M CONCHA
Program Manager

William K McQuay

WILLIAM K McQUAY, Chief
Avionics Simulation Technology Branch
Avionics Directorate

Stanley E Wagner

STANLEY E WAGNER, Chief
System Concepts & Simulation Division
Avionics Directorate

Do not return copies of this report unless contractual obligations or notice on a specific document requires its return.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	January 1997	Final 09/01/93 – 09/01/96

4. TITLE AND SUBTITLE

CAD Tools for the Development and Reuse of Models of Signal Processing Software and Hardware Volume I - Final Report

5. FUNDING NUMBERS

C F33615-93-C-1310
PE 63739
PR A268
TA 02
WU 09

6. AUTHOR(S)

G. A. Frank, and B. E. Clark

**F. G. Gray, J. R. Armstrong,

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Center for Digital Systems Engineering
Research Triangle Institute
3040 Cornwallis Rd.
Research Triangle Park, NC 27709

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Information Directorate
Air Force Research Laboratory
Air Force Materiel Command
Wright-Patterson AFB OH 45433-7623
POC: Louis Concha, AFRL/IFSD, 937-255-1902, ext 3578

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

WL-TR-96-1142

11. SUPPLEMENTARY NOTES

**Virginia Polytechnic Institute and State University
Blacksburg, VA 24061

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

As part of the Rapid Prototyping of Application Specific Signal Processors (RASSP) programming, proof of concept/prototype toolsets were developed to automate:

- Adaptation of signal processing algorithm data flow graphs to fit different hardware architectures
- Creation of high-level VHDL test benches that can be reused as a signal processor evolves

The report describes the toolsets and their construction. Volume II contains user manuals for the toolsets.

14. SUBJECT TERMS

Signal processing, modeling, simulation, VHDL, signal processor, test bench

15. NUMBER OF PAGES 94

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

Table of Contents

1 Introduction 1

1.1 Project Objectives 1

1.2 Team and Roles 2

1.3 Technology Transfer Results 2

2. Overview of the RASSP System Developed for this Project 5

2.1 The Algorithm Partitioning Tool 5

2.1.1 Using the APT System for HW/SW Codesign 5

2.1.2 Tool Components of APT 7

2.1.2.1 The Algorithm Capture Tool of APT 8

2.1.2.2 Hardware Schematic Capture Tools of APT 9

2.1.2.3 The Architecture Sizing and Algorithm Mapping Tools of APT 10

2.1.2.4 Performance Simulation Tools 11

2.1.3 APT Libraries 12

2.1.4 APT Interfaces 14

2.1.4.1 Algorithm Model to Spreadsheet Interface 15

2.1.4.2 Hardware Models to Spreadsheet Interfaces 15

2.1.4.3 Spreadsheet to Performance Models Interfaces 16

2.2 Test Bench Generator 17

2.2.1 Introduction 17

2.2.2 Problem Statement 18

2.2.2.1 Complexity of Test Bench Generation 19

2.2.2.2 Linkage of Test Benches to System Specifications 19

2.2.3 Test Bench Concepts 19

2.2.3.1 A Basic Test Bench 19

2.2.3.2 Test Bench Configurations 21
2.2.3.3 VHDL Test Benches 22
2.2.3.4 Behavioral Test Generation 23
2.2.4 Approach to Test Bench Development 24
2.2.5 Modeling 26
2.2.5.1 Application Area 1: IRST 26
2.2.5.2 IRST Model Development With Ilogix Express VHDL 28
2.2.5.3 Model Complexity And Efficiency 32
2.2.5.4 Modeling SAR 33
2.2.5.5 Library Based Model Development 39
2.2.5.6 Model Simulation Efficiency Studies 41
2.2.5.7 Domain Specific Environment Modeling Software 42
2.2.6 Linkage to System Requirements 44
2.2.6.1 Specification Repository 45
2.2.6.2 Requirements Interfaces 46
2.2.7 Test Plan 49
2.2.7.1 Iterative Test Mode 51
2.2.8 System Integration 54
2.2.8.1 Systems Integration Work 55
2.2.8.2 Demonstration System 57
2.2.9 Conclusions 63
3. Accomplishments 64
3.1 Tool Construction 64
3.1.1 Algorithm Partitioning Tool 64
3.1.1.1 Selection of Tools and Available Libraries 64
3.1.1.2 Development of Target Models 65
3.1.1.3 Development of Interfaces to Generate Target Models 67

3.1.2 Test Bench Generator 70

3.1.2.1 Use of Commercial Tools to Generate Test Benches 70

3.1.2.2 Requirements Captures and Use in Test Benches 70

3.1.2.3 Use of Configuration Declarations to Reuse Test Bench Components. 71

3.2 Library Construction 71

3.2.1 Conversion of SPW Test Benches to VHDL 71

3.2.2 Use of IRAMP Data Bases 71

3.2.3 IRTOOL Models 71

3.2.4 xpatch Models 71

3.2.5 Use of Honeywell/Omniview Performance Model Library 71

3.2.5.1 Packages Subdirectory (apt/data/PML_02a/packages) 72

3.2.5.2 Processor Subdirectory 73

3.2.5.3 Leaf_cells Subdirectory 73

3.2.5.4 Compilation (Analysis) of Library Components for VHDL Simulation 74

3.2.5.5 Compilation (Analysis) of the Library Components for VTIP Use 75

3.2.6 Use of SPW Libraries to Generate Performance Models 75

3.2.7 Development of VHDL DSP Library of Primitives and Applications 76

3.3 Applications of Test Benches 77

3.3.1 SAR Test Benches 77

3.3.2 IRST Test Benches 77

3.3.3 SAR Performance Modeling 77

3.3.3.1 SPW SAR Algorithm Model 78

3.3.3.2 Spreadsheet Hardware Tradeoffs 78

3.3.3.3 VHDL Performance Models 79

3.3.3.4 BONeS Performance Model 80

3.3.3.5 Network II.5 Performance Models 81

3.3.4 JAST Performance Modeling 82

3.3.4.1 Tradeoff Space Explored 82

3.3.4.2 Use of Spreadsheet to Narrow the Design Space 83

3.3.4.3 Use of Performance Model to Identify Bottlenecks 83

4. Future Directions 85

5. Lessons Learned 88

6. Published Papers 89

7. Masters Thesis 90

References 91

List of Figures

Figure 1. RASSP Technology Transfer 3

Figure 2: Algorithm Partitioning Tool and Interfaces 5

Figure 3: COTS Components of the Algorithm Partitioning Tool 8

Figure 4. Dataflow in the Toolset 14

Figure 5. A Basic Test Bench 20

Figure 6. Off-Line Test Bench Configuration 21

Figure 7. On-Line Test Bench Configuration 22

Figure 8. Basic Approach to Test Bench Development 25

Figure 9. Physical Structure of an IRST Sensor 27

Figure 10. Illustration of IRST Algorithm 27

Figure 11. RASSP Stimulus Generator 28

Figure 12 IRST - Overall Statechart 29

Figure 13. IRST - Initialization State 30

Figure 14. RUN_O State Computed Target Position 30

Figure 15. RUN_1 State - Target Position Read From A File 31

Figure 16. Express VHDL Code Template Example 31

Figure 17. Code Efficiency Test Case 32

Figure 18. Deramp Compression Processing 34

Figure 19. SPW Real Number Model 35

Figure 20. Transmitted Signal 36

Figure 21. Downconverted Received Signal 36

Figure 22. FFT of Deramped Signal 37

Figure 23. Real Number Model SPW To VHDL Interface 38

Figure 24. Activity Chart Model of SAR 39

Figure 25. Structural Test Bench for IRST 40

Figure 26. SAR Structural Test Bench 41

Figure 27. IRST Input Frames 43

Figure 28. SAR Radar Return From Two Point Targets 44

Figure 29. Changing An IRST System Parameter 45

Figure 30. SAR System Schematic Diagram 46

Figure 31. Requirements Interface for IRST 47

Figure 32. Math Model for IRST Requirements Interface 47

Figure 33. Math Model for the Requirements Interface (Cont'd) 48

Figure 34. Requirements Interface for SAR 48

Figure 35. Math Model for SAR Requirements Interface 49

Figure 36. A Test Case of Test Group II 50

Figure 37. A Test Case of Test Group III 50

Figure 38. A Test Case of Test Group IV 51

Figure 39 SAR Test Groups of Iterative Test Mode 52

Figure 40. Results of Iterative Mode for SAR Test Group II 52

Figure 41. Results of Iterative Mode for SAR Test Group III 53

Figure 42. SAR Test Group IV Evaluation of Noise Sensitivity 53

Figure 43. Results of Iterative Mode for IRST 53

Figure 44. A High-Level Test Bench Generation System with Test Plan Interface 54

Figure 45. Test Bench Generation System 56

Figure 46. The VHDL Test Simulation Controller 56

Figure 47. User Interface Menu Structure for IRST 58

Figure 48. Test Displays: Test Vectors Used in the File I/O Test Case 58

Figure 49. Test Displays: MUT Outputs for the File I/O Test Case 59

Figure 50. Test Displays: Comparator Outputs for the File I/O Test Case 59

Figure 51. SAR Simulation Result Using File I/O Mode 60

Figure 52. SAR TBGUI Base Window Showing Code Generated Mode and Parameter Entering 61

Figure 53. Code Generated Mode of Test Group II: Range of Multiple Targets 62

Figure 54. Target Report 62

Figure 55. Top-Level Graph in SPW 78

Figure 56. Top-Level Model of RASSP Benchmark Algorithm 80

Figure 57. Range Compression Diagram 81

Figure 58. Network Model 82

1 Introduction

1.1 Project Objectives

This project has two objectives:

1. Reduce the cost and risk of model year upgrades by automating the process for adapting signal processing algorithms to different signal processor architectures.

   The ability to adapt legacy software for signal processors to operate on new hardware configurations is an important part of the model-year concept. Partitioning and allocation are essential steps in successfully adapting signal processing software to different hardware architectures to obtain maximum performance and improved system reliability. Our approach concentrated on the use of representation transformations in order to adapt signal processing algorithm data flow graphs to fit different hardware architectures. Automatic transformation of these algorithms can improve productivity and reduce the number of induced design errors. The automation of the partitioning process also provides the potential for increasing the reusability of libraries of high-performance signal processing algorithms.

2. Reduce the cost and risk of verifying VHDL virtual prototypes of signal processing systems by automating the process for generating high-level test benches.

   Our approach focused on automating the creation of high-level VHDL test benches that can be reused as a signal processor evolves through its model-year upgrades. During this evolution, system requirements will change. Consistent with the RASSP concept of virtual prototyping, we constructed high-level VHDL test benches that captured the essential system requirements in ways that are easy to change as the system evolves.

RASSP applications were employed as driving benchmark problems for tool design and test.

1.2 Team and Roles

The Research Triangle Institute (RTI) and Virginia Polytechnic Institute and State University (Virginia Tech) were teamed in this project. The RTI team, under the leadership of Dr. G. A. Frank, brought over ten years of experience in signal processor architecture performance evaluation and CAD tool development. In addition to overall responsibility for the program, the RTI team was responsible for development of the Algorithm Partitioning tool (APT), its interfaces and its libraries.

The Virginia Tech team, under the leadership of Dr. J. R. Armstrong and Dr. F. G. Gray, brought extensive experience with VHDL, hardware design at the chip and system levels, and system performance design and assessment. The Virginia Tech team was responsible for development of the test bench generation tool and for the VHDL interfaces of the APT.

Captain Edmund Gieske and Lieutenant Richard G. Bishop of Wright Laboratory were the project monitors.

1.3 Technology Transfer Results

During the contract, we have established a working relationship with Northrop-Grumman. They used APT toolset to model the signal processing component of their JAST avionics architecture. The trade studies conducted with the APT tools were the primary component of their benchmarking and resource utilization report. Edited versions of their descriptions of the tool set and of the analysis results were included in Volume 2 of the second report, titled "APT Tool Description" and "Edited Extracts from the Northrup-Grumman JAST Avionics Benchmarking Report," respectively. The below paragraph and figure are extracted from the Northrup-Grumman report:

The benchmarking and resource utilization task results were derived from the process and procedures developed in the RASSP program. The first step was to evaluate the RASSP benchmark algorithms using the JAST hardware concepts as shown in Figure 1. The MIT strip SAR algorithm was used for this purpose. This approach accomplished several things. First, the analysis technology and procedures used in the RASSP program were transferred to the JAST program. Second, the RASSP benchmarking algorithm provides a quantifiable comparison of the performance of the RASSP hardware (current technology) with the projected JAST hardware. Third, a high confidence estimate of the JAST resources required to perform the new radar algorithms were obtained by using the verified and validated analysis approach developed in the RASSP program.

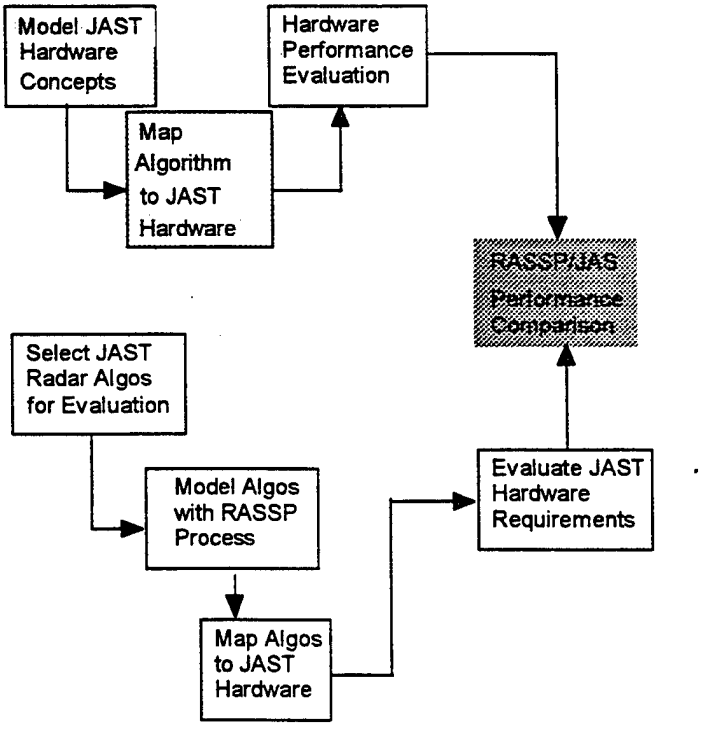


Figure 1. RASSP Technology Transfer

We have had close cooperation in our tool development from CACI, the vendors of the Network II.5 performance modeling tool, and Mercury Computer Systems, Inc. We have also shared models with Cadence Alta Group, the vendors of SPW.

Our test bench generator work has led to several publications and a tutorial on test bench development at the Fall 1996VHDL International Users Forum. The publications and Master Theses which resulted from this project are listed in Sections 6 and 7.

2. Overview of the RASSP System Developed for this Project

2.1 The Algorithm Partitioning Tool

The Algorithm Partitioning Tool (APT) is designed to facilitate the transition from functional algorithm description and VHDL or schematic hardware architecture and component description to a dynamic performance model of the algorithm partitioned and mapped onto the architecture.

2.1.1 Using the APT System for HW/SW Codesign

Codesign is done by using APT with a five-step process that is illustrated in Figure 2.

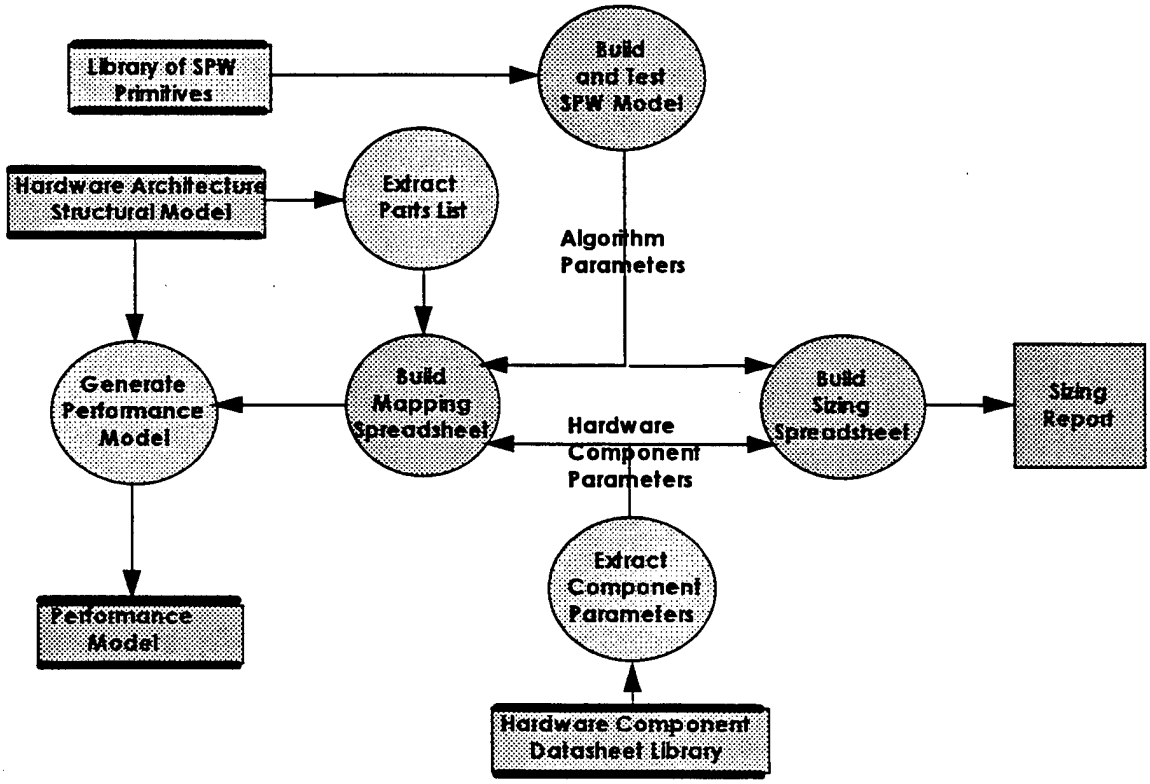


Figure 2: Algorithm Partitioning Tool and Interfaces

1. A functional algorithm model is built and tested using SPW. These models are extensively parameterized to allow rapid functional verification on small test sets. These models are also built hierarchically so that they can be tested in a bottom-up fashion and so that components can be reused in different algorithms.
2. The sizing spreadsheet is used to roughly determine the number of components required and to identify the critical segments of the algorithm. The sizing spreadsheet is a valuable tool for preliminary tradeoffs. Sizing spreadsheets are used to determine processor and memory requirements (or required algorithm modifications) and to compare possible processor instruction set and clock rate assumptions. The APT system generates the sizing spreadsheet automatically from information extracted from the SPW functional model and the processor characteristic library. The user allocates all or part of the algorithm functions to any of the candidate DSPs, and the spreadsheet makes a static calculation of the processor utilization. If the utilization is greater than 100%, then the sizing spreadsheet indicates that multiple processors are required to execute the part of the algorithm allocated to the processor. Similarly, the user allocates the memory requirements for the algorithm functions to memories in the system, and the spreadsheet makes a static calculation of the memory occupancy.
3. Once an appropriate system configuration has been selected based on sizing analysis, a hardware architectural model is constructed. This model is built using either a VHDL structural model with components from an augmented Honeywell/Omniview Performance Modeling Library (PML) or the Network II.5 graphics interface, netgen.
4. Next, a mapping spreadsheet is constructed from the algorithm model developed in Step 1, and the hardware model developed in Step 3. APT automatically generates the spreadsheet from information extracted from the SPW algorithm and either the VHDL or the Network II.5 hardware model. The user then maps the algorithm functions to processing and memory components in the hardware model. If an algorithm function is mapped to multiple processors, then the SPW single execution thread is converted into a multi-thread version for faster execution.

5. When all algorithm functions have had their processing requirements mapped to processors and all their memory requirements mapped to memories, and the utilization of all components is satisfactory, (based on a static model), a performance model can be generated. Simulation of the performance model provides memory and interconnection utilization information, as well as dynamic and more accurate processor utilization information.

2.1.2 Tool Components of APT

As shown in Figure 3, APT consists of libraries and interfaces that surround six Commercial Off-The-Shelf (COTS) tools: an algorithm design tool (SPW), a spreadsheet (2020), a VHDL simulator (Optium 5.2, by Viewlogic, previously Vantage 5.2), a VHDL parser (VHDL Tool Integration Platform (VTIP) from CAD Language Systems, Inc.), an enhanced library of performance models (Performance Model Library (PML_02a) by Honeywell/Omniview), and a performance simulation tool (Network II.5). APT uses the Signal Processing WorkSystem (SPW) of the Cadence ALTA Group to capture the algorithm and verify its functionality. APT captures hardware component characteristics from VHDL processor characteristic files. These characteristic files can be generated by APT from the PML_02a library components using the VTIP VHDL parser or can be generated by hand from spec sheets. APT captures hardware component connectivity from either a VHDL structural model or from a Network II.5 generated schematic. APT uses the 2020 spreadsheet from Access Technology for sizing an architecture from algorithm requirements, and for partitioning and mapping an algorithm onto a specific architecture. Once a mapping spreadsheet has been generated, APT generates a VHDL or Network II.5 performance model from the spreadsheet output and the component connectivity data. APT uses the Network II.5 performance simulator from CACI Products Company or the Optium 5.2 VHDL simulator from Viewlogic to perform a dynamic simulation.

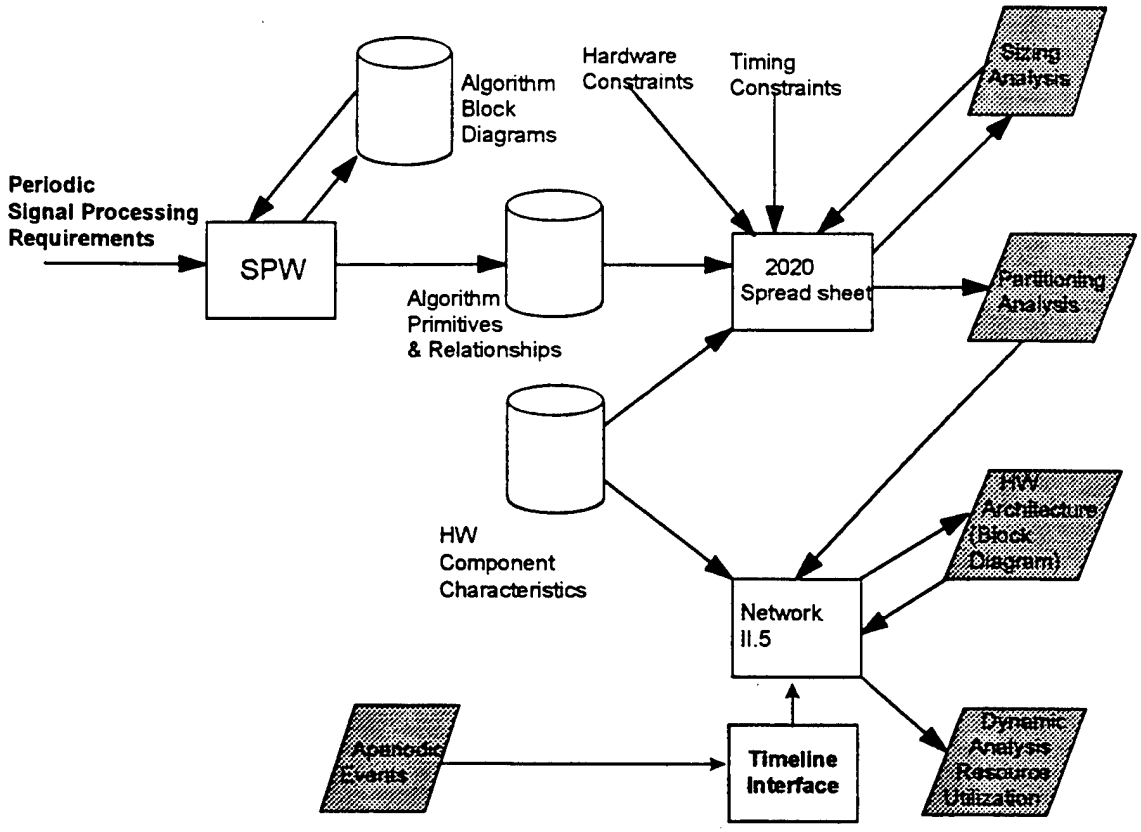


Figure 3: COTS Components of the Algorithm Partitioning Tool

2.1.2.1 The Algorithm Capture Tool of APT

SPW is a software package for developing, simulating, debugging, and evaluating digital signal processing (DSP) and communications system algorithms. SPW provides libraries of DSP and communications primitive "blocks" that can be combined in the block diagram editor (BDE) to represent an algorithm. The BDE functions hierarchically with a symbol at the higher level representing an entire schematic or sub hierarchy of schematics. Properly constructed models can be simulated, using the Signal Flow Simulator. With other components of the SPW an algorithm can be thoroughly tested for functional correctness. In addition to the built in simulator, the system provides a code generation system (CGS) that will generate C code from the schematic targeted to the host machine or various signal processors from TI, Motorola, or AT&T. The Cadence Alta Group provides a Tool Interface Language (TIL) with the

SPW package. TIL can be used to develop user-specified custom tools accessing the BDE database. The Signal Flow Simulator and CGS are both TIL applications.

The APT tools employ TIL programs to capture algorithm size and dataflow characteristics from a BDE database (model of the algorithm). This, combined with control flow data from the simulator or CGS, provides the workload characterization necessary to map the algorithm onto a hardware architecture and conduct both static and dynamic performance analyses. An algorithm designer employs SPW to graphically capture and functionally validate the design of a signal processing algorithm. If the SPW model is to be used for performance analysis, it must be constructed (at the "leaf node" level) from primitives that have the RTI developed extract TIL programs attached. Currently, a subset of the standard libraries (sufficient for the DSP applications being investigated) included in a special library, "apt_lib", are employed for this purpose.

2.1.2.2 Hardware Schematic Capture Tools of APT

APT uses two forms of input to describe the hardware:

- A processor characteristic library that is used to capture information about the instruction sets and clock rates of processors, input devices (such as sensors), and output devices (such as displays); and
- A structural model of the components, their interconnects, and their parameters (such as processor type). A structural model is composed of components selected from five types: input devices, output devices, storage devices (e.g., memories), processors, transfer devices (e.g., buses), and gateways (which represent nodes in multistage interconnection networks).

The structural model also describes the interconnections among the components.

APT uses either a VHDL structural model with components from the enhanced PML_02a library or the schematic capture tool netgen, which is part of the Network II.5 tool set from CACI, to represent

structural information. The enhanced PML_02a library and the Network II.5 library both contain a basic set of components, which include processors and memories. In the VHDL structural model, a bus is represented by a signal and a set of Bus Interface Units (BIUs), with one BIU for each device on the bus. Network II.5 treats the bus and the BIUs as a single component. We have developed a hierarchical approach to capture the BIUs attached to a common signal so that the BIUs and their signal are represented as a single component in the top-level structural model. This approach also ensures that all the BIUs connected to a common signal receive the same generics.

2.1.2.3 The Architecture Sizing and Algorithm Mapping Tools of APT

APT uses the 2020 spreadsheet from Access Technology to develop gross sizing budgets and to analyze alternative partitionings of the algorithm onto the architecture. This spreadsheet provides a facility for creating and modifying spreadsheets with externally produced ASCII files. The spreadsheet is employed in two modes which are referred to as "sizing" and "mapping".

Sizing mode is employed in early trade studies when architectures have not yet been defined. Sizing mode uses a static performance analysis to address issues such as how many processors of a given type and how much memory are required to execute a primitive, a section of the algorithm, or the entire algorithm within the allotted time. In sizing mode, two performance metrics are calculated: average processor utilization, and maximum memory occupancy.

For initial trades where one is comparing different processors, memory requirements, and comparing alternative partitioning strategies a much smaller spreadsheet suffices and is desirable. A sizing analysis of the RASSP SAR algorithm on a Mercury Raceway architecture produced a spreadsheet where the 30 processors are represented by 4 columns for: (a) range compression processors, (b) corner turn processors, (c) azimuth compression processors, and (d) image formatting processors.

Mapping mode is employed to perform static analysis of a specific architecture, to create alternative mappings onto a specific architecture, and to create performance models for dynamic analysis of an algorithm/architecture configuration. During mapping, the same static performance metrics are used as in sizing analysis: average processor utilization and maximum memory occupancy. In this mode, spreadsheet output is used to produce a dynamic performance simulation model that can be used to assess additional issues such as latency and resource contention. The same algorithm data is used for both modes, but different hardware descriptions are used. In the mapping mode, a column for each processor and two columns for each memory are required. The processor and algorithm descriptions are stored in separate input files. Thus, any algorithm in the user's library can be mapped or sized with any hardware architecture.

2.1.2.4 Performance Simulation Tools

The APT employs either Optium 5.2 or Network II.5 for performance simulation. One of the main considerations in the design of a computer system is the effect of conflicts over access to computing resources. These effects are not readily determined by the use of analytical models. Because both VHDL simulators and Network II.5 model the interactions between all devices in the system, the effects of resource conflicts are identified.

The PML_02a library contains VHDL packages that monitor performance parameters and display the information in graphical format that is easy to read.

Network II.5 supports separate representations of hardware and software, making it possible to reuse hardware models in the evaluation of multiple algorithms on the same architecture. Network II.5 provides an extensive statistical report on each simulation, and also generates strip charts of the utilization of components over time. Network II.5 also provides animation of a simulation, so that the user can visualize the behavior of the computer system being modeled.

2.1.3 APT Libraries

The Algorithm Partitioning Tool (APT) libraries consist of:

- A library of SPW primitives designed for extraction of algorithm characteristics from the SPW functional model. This library is delivered in a SPW dump file, apt.dmp. When retrieved it will reside in SPW as the library, apt_lib. This is accompanied by a library of TIL source files in the directory $aptsys/til which can be used to recompile, if necessary, the extraction programs.
- A library of 2020 command files for constructing a spreadsheet that can be used either for static analysis of resources required by the algorithm or for mapping the algorithm onto a specific architecture. This library resides, along with command files for support of sizing and mapping, in the apt/tools/cmds directory.
- A library of processor characteristics in the form of 2020 command files. This library has been extracted from the PML library and processor datasheets. APT includes tools to update this library as additional processors are added to the PML library. APT also allows processor characteristics to be created manually (e.g., for notional architectures) and added to the library. This library resides in the apt/data/proc directory. Processors currently characterized in the library are shown in Table 1.

Table 1: Digital Signal Processors in the Processor Characteristics Library

Manufacturer	Processor Identifier
Analog Devices	SHARC 21062
Hughes	DSPE
Intel	i860
Intel	i960MX
Intel	i386
Intel	Pentium

Motorola	96002
Texas Instruments	TMS320C40
Texas Instruments	TMS320C80
AMD	29050
MIPS	R4000

- Four SPW libraries of benchmark algorithms, apt_3_pol and apt_1_pol, the RASSP benchmark I and a single polarity version respectively, apt_rbgm, a real beam ground map algorithm modeled under a JAST support contract, and test_sys, a small test system using the functions of the RASSP benchmark I algorithm with small vectors that was used to verify the SPW modeling with apt_lib blocks and for initial interface development. These libraries are delivered in the SPW dump file apt.dmp along with apt_lib. The SPW algorithm models are extensively parameterized so that by changing a few parameters on the top level diagram one can automatically adjust data structure sizes, number of subswaths, FFT sizes, decimation rates, FIR filter sizes, etc. This parameterization allows rapid tradeoffs between functional characteristics and computational resources. Data sets extracted from these models are included in the apt/data/sw directory. These data sets can automatically produce sizing or mapping spreadsheets for any architecture represented in the apt/data/hw directory.

- A library containing the VHDL structural model (and derived performance model with the test_sys algorithm mapped onto it), contained in the models/vhdl/vpirace16 directory, and a library of data sets extracted from this model and from netgen schematics, contained in the apt/data/hw directory.

- An enhanced library of performance modeling components suitable for use in VHDL structural performance models. The Honeywell PML_02a component library was enhanced by adding additional components. Also, some of the leaf cells were modified for this application. This library is delivered in the apt/data/PML_02a directory.

2.1.4 APT Interfaces

The Algorithm Partitioning Tool includes several interfaces that support the transformation of data from one COTS tool representation to another. Figure 4 shows the flow through and between COTS tools and interfaces.

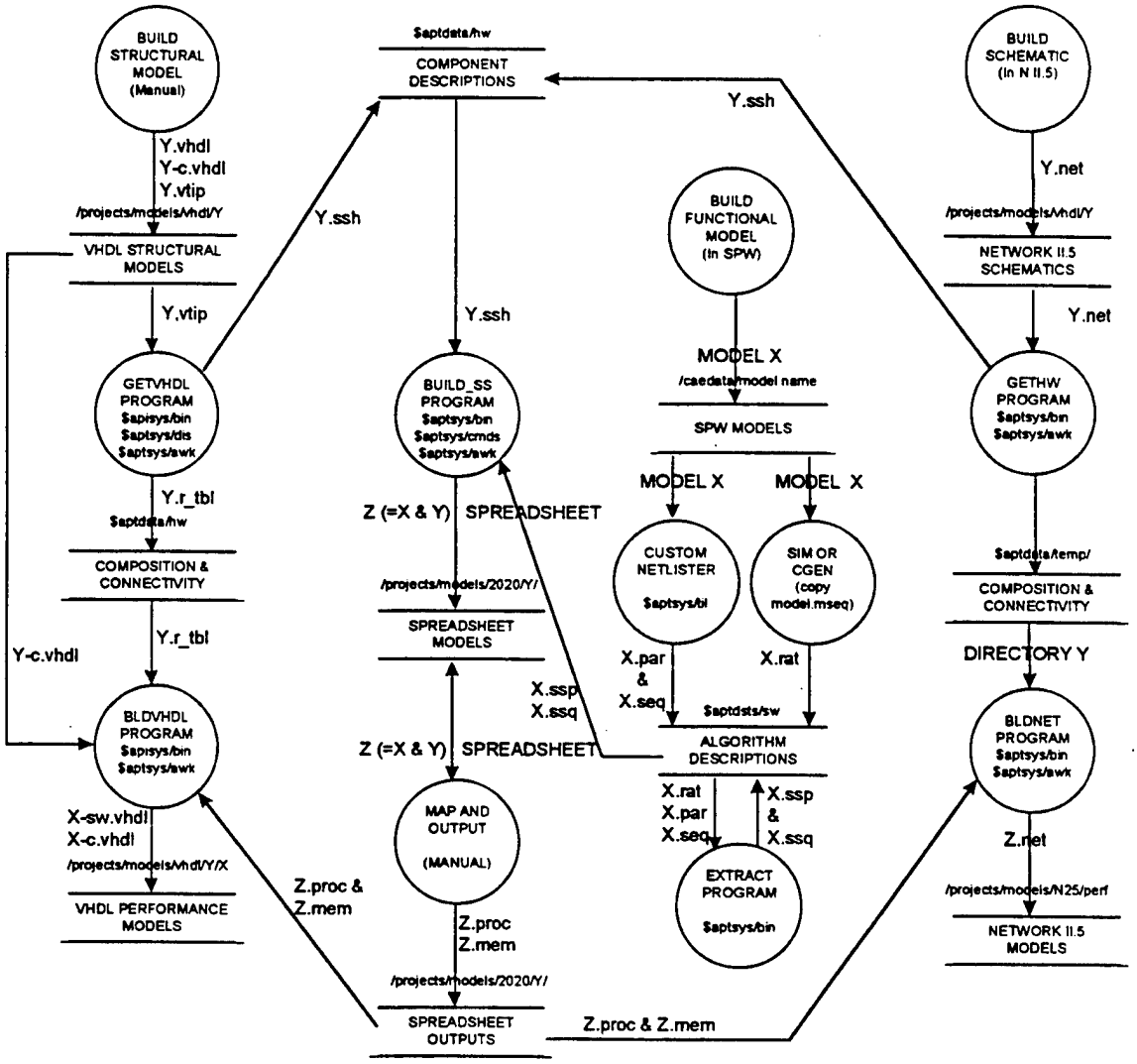


Figure 4. Dataflow in the Toolset

2.1.4.1 Algorithm Model to Spreadsheet Interface

The next to the right hand column along with the "BUILD_SS PROGRAM" bubble of Figure 4 shows the interface from the algorithm model to the spreadsheet. Having built the model in SPW using the apt_lib library, the user runs the custom netlister using the master tool, extract, from apt_lib and executes CGEN. This produces three data files with a common name and suffixes of par (parameters), seq (sequencing), and rat (rates). A call to the C program extract reads these three files and produces two shell scripts with the same name and suffixes of ssp (parameters) and ssq (data-flow predecessor and successor relationships). These two files are called by the build_ss spreadsheet building script to insert into the 2020 command file the calls that will produce the function rows and predecessor/successor data.

Instructions are provided in the User Guide for manual production of the ssp and ssq files. Thus, the user can produce a spreadsheet for sizing or mapping without building a SPW model. This procedure will produce a performance model from spreadsheet output if the architecture data was extracted from a structural model or schematic.

2.1.4.2 Hardware Models to Spreadsheet Interfaces

The top two bubbles in the two outside columns in Figure 4 show the interface from the hardware models to the spreadsheet. Both gethw and getvhdl produce a shell script with a suffix of ssh. This file is called by the build_ss spreadsheet building script to insert into the 2020 command file the calls that will produce the processor and memory columns. Both extraction programs also produce composition and connectivity data that are used with the spreadsheet output to build the performance model.

The script getvhdl analyzes the partially configured structural model using VTIP. It then calls a C program, acet, that produces the .ssh shell script for spreadsheet construction. This is followed by a call to the C program conet that produces an interim "connectivity" file. Getvhdl then calls the shell script

Bld_Route which employs a series of three awk scripts to produce routing tables for the VHDL performance model. This file, with suffix r_tbl, is stored in the apt/data/hw directory.

Another script, get_proc, calls the C program, pcet, which uses VTIP to extract the necessary information from the PML models and produces the 2020 command file for a processor in the apt/data/proc directory.

The shell script gethw calls an awk script that produces the .ssh shell script for spreadsheet construction and decomposes the Network II.5 schematic into a subdirectory of component files in the apt/data/temp directory.

Instructions are provided in the User Guide for manual production of the ssh file. Thus, the user can produce a spreadsheet for sizing or for static analysis of a mapping without building a structural model or schematic. This procedure will not provide sufficient architecture data to produce a performance model.

2.1.4.3 Spreadsheet to Performance Models Interfaces

The bottom bubbles on the two outside columns in Figure 4 show the interfaces from the spreadsheet to the performance models. Both the bldvhdl and bldnet scripts take the two spreadsheet output files as input.

The bldvhdl script also employs a standardized vhdl configuration file and the r_table routing file produced by getvhdl. It calls two nawk scripts:

bv_pass1.awk translates Intermediate Form into VHDL application sw

bv_pass2.awk creates a configuration file for the VHDL performance model

The bldnet script also employs the files in the temp directory produced by gethw and calls a series of 6 awk scripts:

mem.awk	processes the <modelname>.mem file and creates an auxiliary file describing the memory mapping.
pass1.awk	generates multiple auxiliary files and a first pass translation of the spreadsheet: <modelname>.mac.
pass2.awk	edits the predecessor and successor information based on the loop structures in the spreadsheet.
pass3.awk	edits the predecessor and successor information based on the parallel loops.
pass4.awk	generates a final translation of the .mac file and the auxiliary files into the .out file, a final intermediate form.
pass5.awk	generates the network II.5 processor instruction sets and the network II.5 modules and files.

2.2 Test Bench Generator

2.2.1 Introduction

In this section of the report, we describe the results of the test bench generation research carried out under the RASSP program. We will show that this research developed methodology for the:

- Rapid development of test benches for VHDL [IEEE88] models of DSP algorithms.
- Use of high-level graphics based design tools to generate test bench VHDL code.
- Use of test bench component libraries which allow construction of a range of test bench configurations.
- Use of requirements capture, requirements interfaces, and test plans to link the test bench to the system specification.

This methodology has been applied to two sensor based systems: 1) Infrared Search and Track (IRST)[Cams93], and 2) Synthetic Aperture Radar (SAR)[ShaG94] and a demonstration test bench generation and simulation system was developed which generates model inputs for these two application domains.

The Rapid Prototyping of Application Specific Signal Processors (RASSP) program [RicM94] is based on the model [1] year concept for hardware development where a design in the field is refined and updated from year to year instead of being fully replaced. In this situation, the system specification can change from year to year. Our test bench/ specification interface is designed to account for these model year changes.

The results of this research are most beneficial to VHDL modeling in the digital signal processing domain. Mathematical models in this domain can be used to generate "normal system" inputs while working at a high level of abstraction. Other domains do not always have this property. However, since a high percentage of ASIC designs are for DSP systems [EurA93], the results are still important. And some of the techniques, e.g., the requirements and test plan interfaces, have wider application.

2.2.2 Problem Statement

The research is intended to solve to problems: 1) Complexity of the test bench generation task, and 2) Linkage of test benches to the system specification.

---

[1] In this specific sentence the word model refers to a version of the hardware, not to a VHDL model. One of course can have a VHDL model of a system for each model year!

2.2.2.1 Complexity of Test Bench Generation

A simulation model must be thoroughly exercised to prove that it exhibits the correct behavior. However, generation of tests for a model is a labor intensive task rivaling that of developing the model itself. Moreover, tests generated manually satisfy no formal definition of completeness. What is required is method of test bench development that can be carried out at a high level of abstraction, e.g., in the mathematical domain of DSP systems, which relieves the modeler of the details of test bench development. One also needs a completeness criterion. We will show below that completeness is achieved by accurate modeling of the environment surrounding the system which the Model Under Test (MUT) represents.

2.2.2.2 Linkage of Test Benches to System Specifications

A major goal of the DOD is to have the test bench directly reflect the system specification. Then if the system model executes correctly in the test bench, one can assert that it satisfies the specification requirements. The RASSP program adds an additional requirement in that the specification may evolve from one model year to another and the test benches for a system model should track this evolution. Test benches that have usefulness through the life of a program are also a key to lowering life cycle support cost. Finally, it is important that specification information be stored in such a way that system engineers have easy access to it. In the section below on linking system requirements, we will discuss how these issues are addressed.

2.2.3 Test Bench Concepts

2.2.3.1 A Basic Test Bench

Figure 5 shows a basic test bench[FraG91]. The Model Under Test (MUT) receives test vectors from a Stimulus Generator which also provides an expected (GOLD) response. The Comparator compares the MUT response with expected response and issues GO/ NO GO signals. A test bench can have two types of

feedback. First the model state can be fed back to the Stimulus Generator to model interaction between these two elements. Second, there can be feedback from the Comparator to the Stimulus Generator which allows adaptive testing, i.e., the results of one test dictate what the next test will be. We will discuss below how we applied adaptive testing.

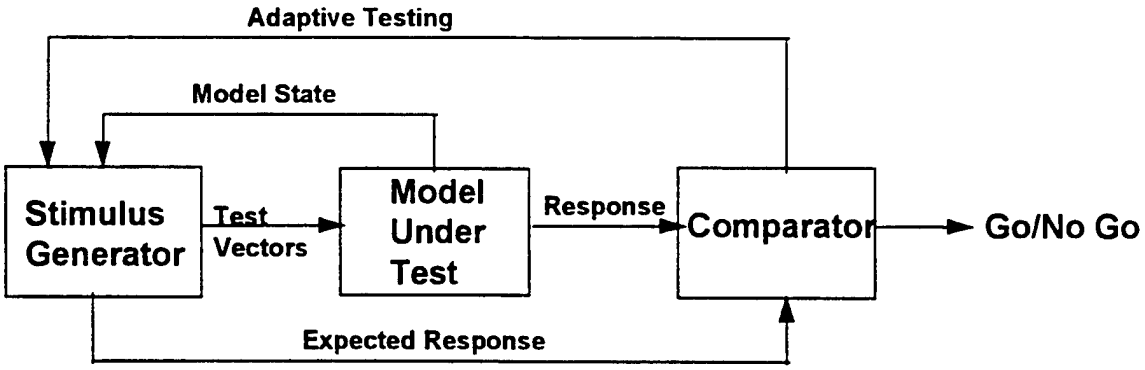


Figure 5. A Basic Test Bench

2.2.3.2 Test Bench Configurations

Test benches can be classified into a number of configurations.

Figure 6 shows an **off-line** configuration, where stimulus data files, generated earlier, are read by the Stimulus Generator and applied to the MUT. The expected response is another data file which is feed to Comparator for comparison with the MUT response. The data files used by the off-line configuration are originally generated by an on-line configuration or can be real system data.

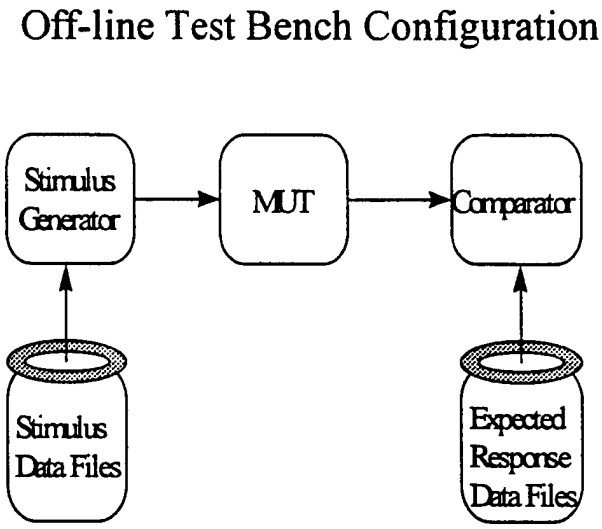


Figure 6. Off-Line Test Bench Configuration

Figure 7 shows an **on-line** test bench configuration. Here the Stimulus Generator is programmed to produce test vectors. The test vectors are fed to a gold model of MUT behavior which generates expected response. The MUT is simulated in the same simulation that generates the test vectors and expected response. In many cases, such as our RASSP work, there is no distinct gold MUT model. Rather, the Stimulus Generator uses the mathematics of the system model to generate the expected response directly.

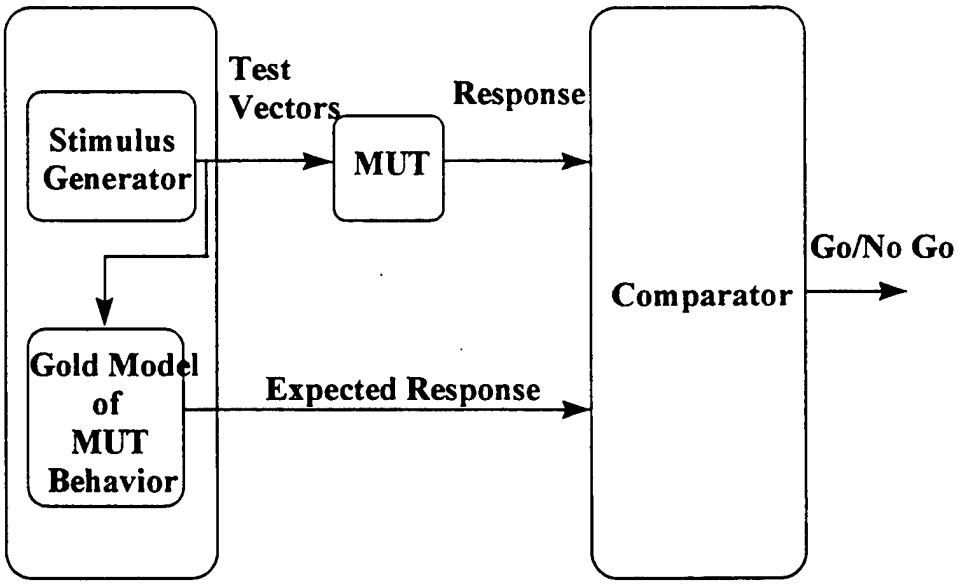


Figure 7. On-Line Test Bench Configuration

Adaptive test benches are on-line test benches which react to feedback from the MUT or the Comparator (See Figure 5) to effect the generation of future test vectors. Below we will discuss how adaptive testing is an important concept in high-level approaches to system testing.

2.2.3.3 VHDL Test Benches

When the Model Under Test (MUT) is a VHDL model, the test bench is also coded in VHDL[ArmJ93]. The MUT is plugged into it and the test bench is the top level component in the model hierarchy, i.e., it is the entity that is simulated. For complicated MUTs, the development of a VHDL test bench is, when left to manual means, a complicated programming task rivaling that of developing the model itself. This problem can be complicated by the possible necessity for the Stimulus Generator to react to feedback from the MUT or Comparator in adaptive testing.

In developing a VHDL test bench, one must develop: 1) A VHDL shell into which to plug the Stimulus Generator, MUT, and Comparator which allows application of test vectors via signal assignment statements or file I/O. 2) The test vectors to be applied by the Stimulus Generator. Task 1 is a

straightforward task and can be easily performed through the use of VHDL structural architectures and configuration bodies. Task 2 consists of test generation for VHDL behavioral models which we discuss next.

2.2.3.4 Behavioral Test Generation

There are two basic approaches to test vector generation, **model based**, and **environment based**. In model based test generation, the model itself is used to perform the test development. It is sometimes referred to as **white box** testing because knowledge of the internal nature of the model is required to develop the tests. Model based testing can be further divided into **model perturbation** and **I/O path sensitizing**. In model perturbation, faults are injected into the model and tests determined which detect these faults. Conventional Automatic Test Pattern Generation (ATPG) at the gate level fits in this category. Researchers at Virginia Tech and elsewhere have applied this approach to behavioral models[ChoC94]. In I/O path sensitizing, one develops tests which activate all paths through the model. No fault model is employed. We have been active in this area of research also, using NSF funding to develop tests based on a Process Model Graph representation of a behavioral model [KapS94, LiW96]. I/O path sensitizing is also a relevant technique for testing conventional software such as C programs.

In environment-based testing, one uses a model of the environment surrounding the system being modeled to develop the tests. In a sense, one is developing normal system inputs. This type of testing is referred to as **black box** testing because one develops these tests without knowledge of the internal structure of the system. For the RASSP project which deals with DSP systems, we have employed environment-based testing because the system inputs are a mathematical function of the environment surrounding the DSP system and could not be derived from the Model Under Test.

2.2.4 Approach to Test Bench Development

Our approach to test bench development is based on the following four principles:

1. Test bench VHDL code elements are initially developed using graphics-based high-level system design tools. These code elements are developed primarily as behavioral models. Using these high-level tools allows one to construct models of the environment based on system mathematics, thus making it easy to establish their correctness.
2. Code elements are refined in a library structure that allows construction of structural models. This is a key element in the RASSP model year concept which allows reuse of test bench code from one year to the next.
3. Environmental Data Generators are used to prepare input files that can be read by the test bench. These data generators are programs specific to the DSP environment we worked with, i.e., IRST and SAR. This software has been integrated into the test bench generation system.
4. Specification values are stored in a repository that is linked to the test bench through a requirements interface.

Figure 8 shows the basic approach to test bench development. The system specification consists of **general requirements and specific requirements**. General requirements specify the class of systems being modeled. For example, a test bench for an IRST system must generate two dimensional arrays of pixels at a fixed clock rate to model the operation of the IRST sensor. Specific requirements select a particular member of the class. For IRST, the specific requirements might be a frame size of 256 X 256, pixel intensity range of 0 to 63, and a frame rate of 10 frames/sec. Below we will show how the specific requirements are stored in a Specification Repository that is easily accessed by system engineers and these values are fed automatically forward to the test bench system.

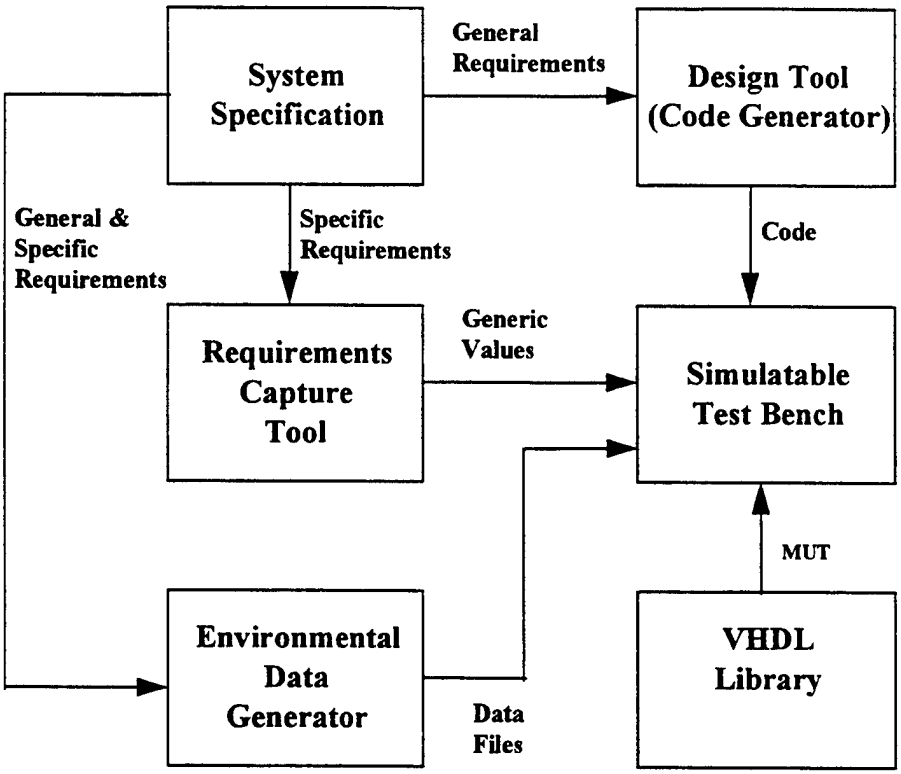


Figure 8. Basic Approach to Test Bench Development

General requirements are input to the design tools used for code generation. The resultant code is general in that it is parameterizable with VHDL generics. The specific requirements are fed to a math model which operates on them to produce **derived specific requirements** that are generic values. As shown in Figure 8, this math model can be implemented in a Requirements Capture Tool such as RD100 [Asc]. This was our original approach. It has the advantage that system managers may already be using such a tool for systems analysis. However, the math model can be implemented just as easily in MATLAB [MAT92], EXCEL, or a C program. Another possibility is to put the math model in the test bench code, but this adversely effects the generality of the test bench code library. Below we will add test plans to the generic value development interface which can override the nominal specific requirements coming from the system specification.

Both general and specific requirements are fed to environmental data generators which develop data files that can be read by the test bench during simulation. The Model Under Test is selected from a VHDL

library. Given these four elements: test bench VHDL code, generic values, data files, and MUT, a test bench is fully assembled.

2.2.5 Modeling

In this section, we describe our approach to modeling. Although the models we developed were for the test bench code, the techniques employed apply to a wide range of modeling situations. Our approach to modeling has two main features:

1. Use high level, graphics based design tools to create initial behavioral test bench models. In the DSP area, the mathematics of the environment is well defined. If we model at a high level, we can concentrate on applying the mathematics to the problem and ignore the language details, thus insuring model correctness. The high level tools have VHDL interfaces which allow us to dump simulatable VHDL code. We used this approach to develop our initial IRST test bench early in the RASSP project and were able to do so in less than a month.
2. Develop a library of primitives which allow the creation of structural models. Here, the code elements originally developed as behavioral models are refined over time in a library structure. The test bench model is then constructed structurally using a schematic capture tool. The library structure supports code reuse which is a key element in RASSP model year design concept.

2.2.5.1 Application Area 1: IRST

Infrared Search and Track (IRST) systems are a class of passive military infrared sensor systems which can reliably detect and track targets that emit an infrared signature [CamS93]. The components of the system consist of an infrared sensor, IRST computer, track files data base, and display hardware. The inputs to the computer developed by the sensor consist of a continuous sequence of infrared image frames of targets superimposed on background clutter. Figure 9 shows an IRST sensor.

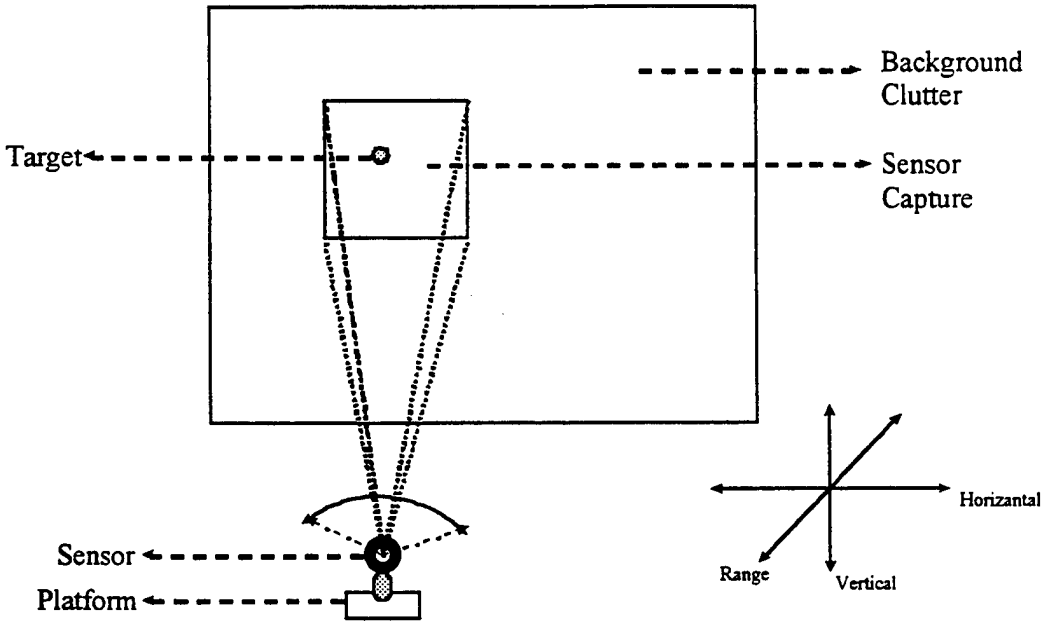


Figure 9. Physical Structure of an IRST Sensor

The function of the IRST MUT which represents the computer is to find the best alignment of successive frames so that if two successive frames are differenced, the result is just the target movement. Figure 10 shows this process. Figure 10 -d shows the result of differencing.

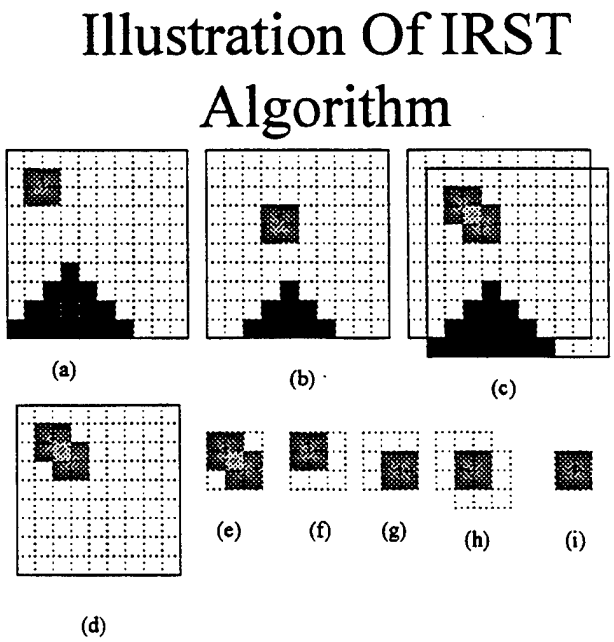


Figure 10. Illustration of IRST Algorithm

Figure 11 shows the IRST stimulus generator model which consists of the following sub models:

1. Target - Using information from a user interface, generates target signature of varying shapes, speed, and direction.
2. Environment - Models the background scene, i.e., the clutter against which the target will be super-imposed.
3. Platform - Models the effect of platform motion, i.e., roll, pitch and yaw.
4. Sensor - Combines the information from the target, environment, and platform to produce the stimulus generator output, i.e., a two-dimensional array of pixels depicting targets on a clutter back ground.

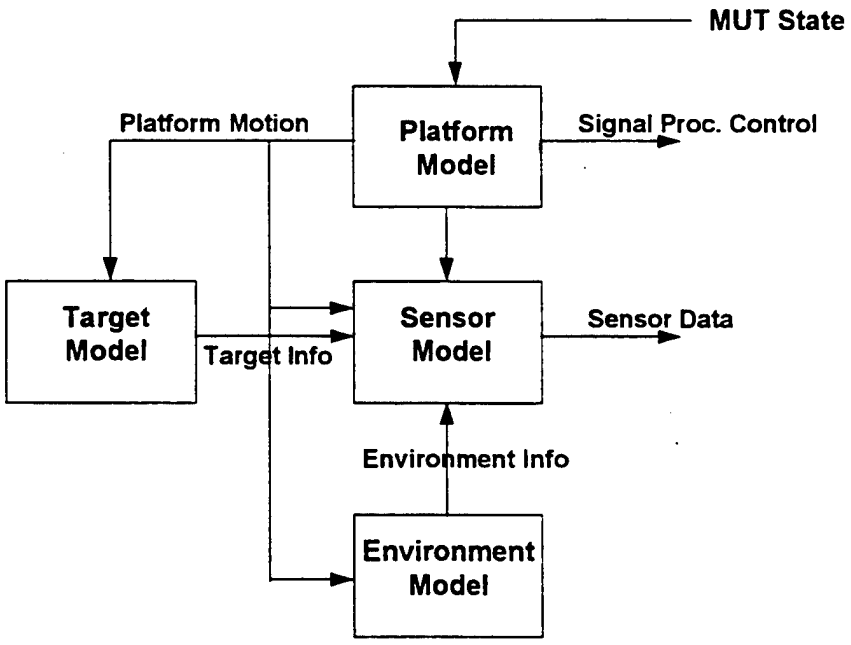


Figure 11. RASSP Stimulus Generator

2.2.5.2 IRST Model Development With Ilogix Express VHDL

Our initial model development for IRST was done using Ilogix Express VHDL [Ilog92], which is a CASE tool that uses state charts [HarD87] and activity charts as the graphical representation. Statecharts are used for control intensive models, activity charts model data flow. The IRST model was done using

statecharts [HriS95,ArmJ94]. Figure 12 shows the top level state chart for IRST. State charts allow for concurrent states. Note that states FUNC and CLOCK are concurrent.

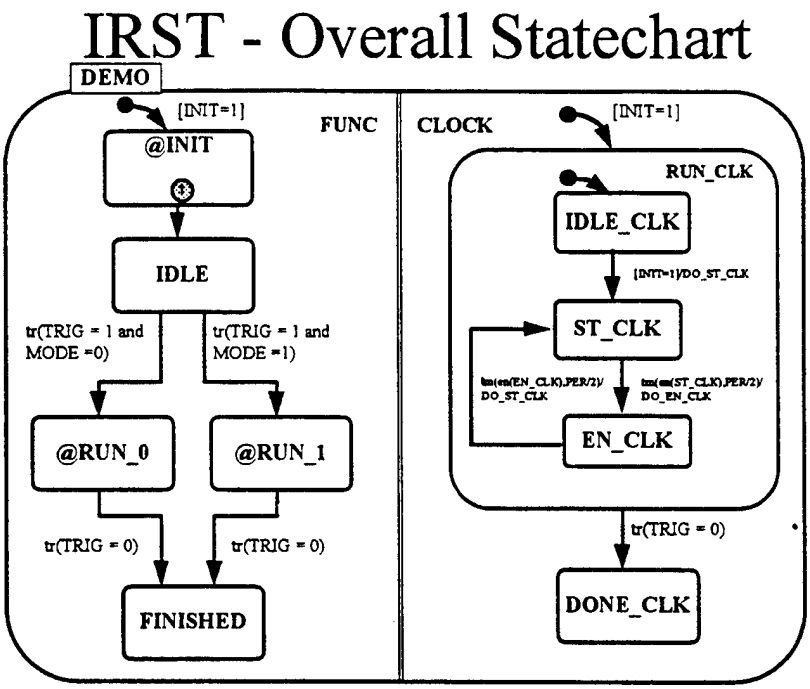


Figure 12 IRST - Overall Statechart

Nesting of states is also allowed. Thus states IDLE_CLK, ST_CLK, and EN_CLK are nested within the state RUN_CLK. State transitions can be triggered by events or elapsed time. States with their names preceded by an @ are super states which imply an underlying state chart. Thus INIT, RUN_0, and RUN_1 are super states. Figures 13, 14, and 15 show the state charts for these super states.

In addition to the state charts, one typically has to fill in code templates which define functions invoked when a transition is made. Figure 16 shows a template for a function DO_ALG_CALC which is invoked when the INIT state chart (Figure 13) is invoked.

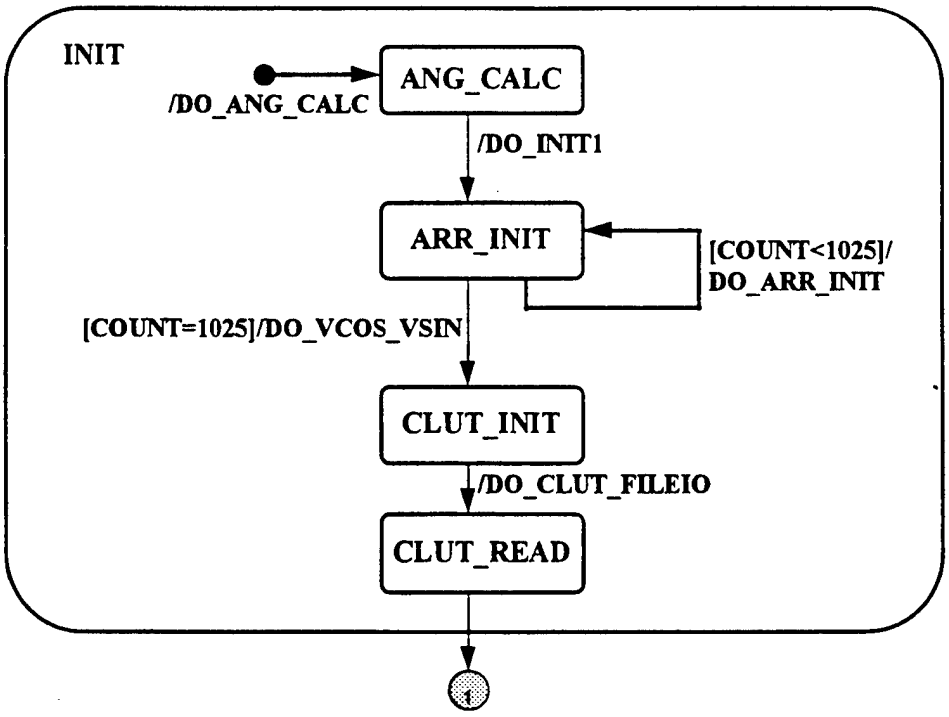


Figure 13. IRST - Initialization State

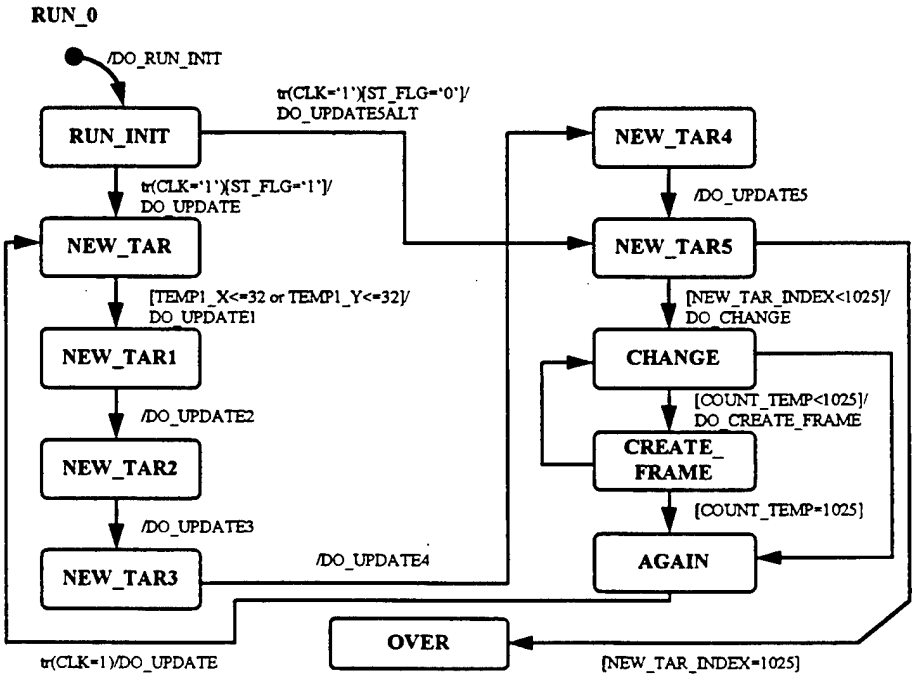


Figure 14. RUN_O State Computed Target Position

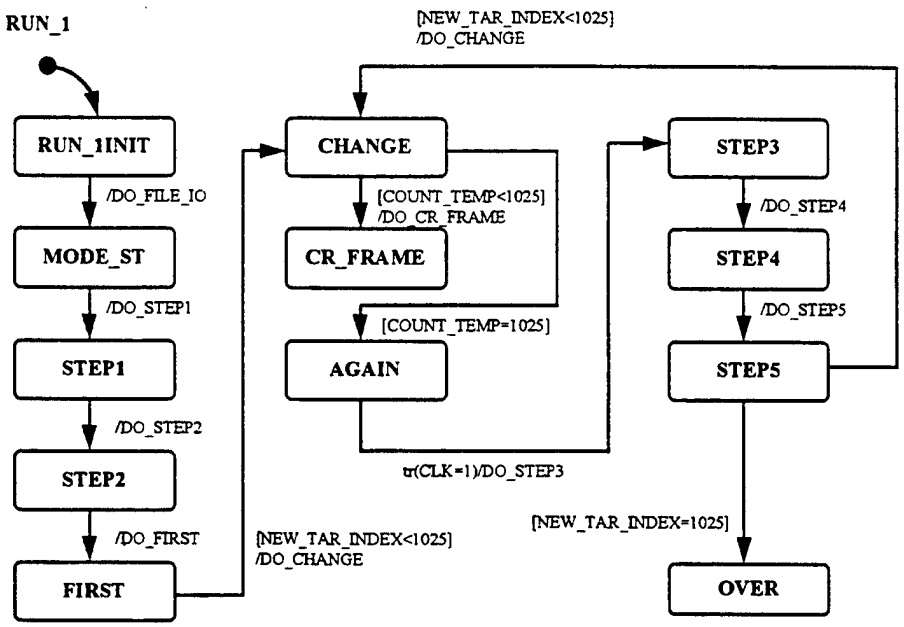


Figure 15. RUN_1 State - Target Position Read From A File

The completed state chart model can be simulated in the graphics mode and when it is found to be correct, C, VHDL or Verilog can be dumped.

```
ACTION DICTIONARY
    Project: IRST

    DO_ANG_CALC
    Defined in chart: INIT
    Definition: TAR_XCOORD:=XCOORD;
                TAR_YCOORD:=YCOORD;
                if (MODE_MOTION=1) then
                  RAD_ANGLE:=0;
                end if;
                if (MODE_MOTION=2) then
                  RAD_ANGLE:=(90*3.14)/180;
                end if;
                if (MODE_MOTION=3) then
                  RAD_ANGLE:=(ANGLE*3.14)/180;
                end if;
                R_VECTOR:=VELOCITY;
```

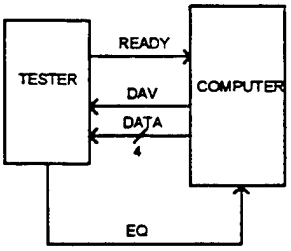
Figure 16. Express VHDL Code Template Example

2.2.5.3 Model Complexity And Efficiency

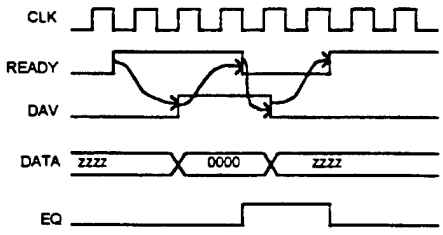
It is important to assess model complexity and model efficiency. The complexity data for the IRST Ilogix model is:

1. State Charts: 43 nodes and 56 arcs.
2. Templates: 520 lines of text.
3. Resultant VHDL(Dumped Automatically): 948 lines.
4. Programming Time: it took us about a month because we were learning the Ilogix software and the IRST specification. Assuming knowledge of both, a model of this complexity could be done in under a week.

The efficiency of Ilogix models was compared with other approaches in a separate study. The test case involved modeling a handshaked exchange between a Computer and a Tester. The computer feeds words to the tester. If three successive words are equal, the tester asserts an EQU output. Figure 17 shows the system block diagram and a timing diagram for one data exchange. We modeled the tester completely, but only the bus interface of the computer was modeled.



Equality Tester Block Diagram



Equality Tester Timing Diagram

Figure 17. Code Efficiency Test Case

The results of the study are shown in Table 2. We compared flat (hand coded) VHDL models with models developed from Ilogix state charts and SPECCHARTS as implemented at UC Irvine [GajD94]. In terms of simulation speed, the hand code is markedly more efficient than the code generated by Ilogix in four different coding styles, and somewhat more efficient than code developed from SPECCHARTS. When synthesized, the Ilogix code required about 20% more cells than the hand code. We present these results not to criticize Ilogix and UC Irvine systems, for they are both very useful for high level modeling, but merely to illustrate that more research needs to be done on the transformations from these graphic representations to VHDL.

Table 2 Model Efficiency Comparison

Style	Simulation Time	Cell Count
Flat(PMG)	.47 sec.	216
SPECCHARTs normal	.60 sec.	Not synthesizable
SPECCHARTs flat	.58 sec.	Not synthesizable
Ilogix Style 1	1.49 sec.	Not synthesizable
Ilogix Style 2	1.49 sec.	Not synthesizable
Ilogix Style 3	1.72 sec.	258
Ilogix Style 4	1.66 sec.	258

Style 1: behavioral/in line, Style 2: behavioral/procedural, Style 3: RTL/in line, and Style 4: RTL/procedural.

In summary, our experience with Ilogix Express VHDL reinforced our belief that development of models with high-level graphics based tools is a very effective approach to model development. It allows the modeler to concentrate on the mathematical structure of the model without worrying about language details. The approach can markedly speed up the model development process.

2.2.5.4 Modeling SAR

In Synthetic Aperture Radar (SAR), an effectively long antenna is achieved by moving the antenna along a line and emitting pulses at regular intervals [XuZ95]. Then through signal processing means, the returns are combined to achieve the effect of an antenna with a wide aperture. The stimulus generator in the SAR model has to perform the following functions [ZueB94] :

1. Generation of the transmitted signal which is a pulse of the form $s = \cos 2\Pi(f_c t + Kt^2/2)$. i.e., a "chirp" is emitted where the frequency is varied linearly over the duration of the pulse.

2. Generation of the received signal which is the transmitted signal delayed by t_0 which is proportional to target range, .i.e. $r = \cos 2\Pi (fc(t - t_0) + K(t - t_0)^2)$

3. Down conversion. Here the value of f_c is shifted down from the GHz to the MHz range.

4. Deramping. The down converted received signal cannot be processed through conventional filtering. Thus it is "deramped" by multiplying it by its complex conjugate. This is better illustrated if the signal is represented in complex form. Figure 18 illustrates the process.

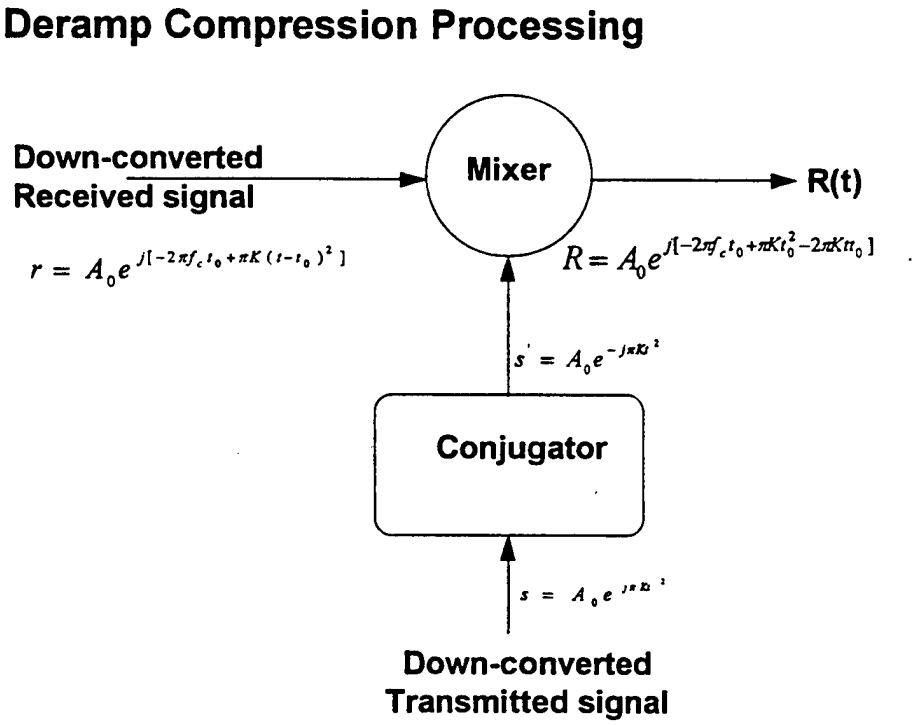


Figure 18. Deramp Compression Processing

In the final result (R) it is instructive to look at the term $-2\Pi f_c t_0 + \Pi K t_0^2 - 2\Pi K t_0 t$. Note that the first two terms are constants, and in the third term the frequency is Kt_0. Thus what we have produced is a monochromatic sine wave whose frequency is proportional to range.

In our SAR test bench work we developed the model of the transmission and return process using Signal Processing Worksystem (SPW) from Cadence [Cad94].

Figure 19 shows the schematic for the SAR test bench. Some of the primitives are basic SPW DSP primitives such as Complex Conj and Z^{-1} (delay). Other symbols such as Gen_chirp imply an underlying schematic which is made up of SPW primitives. Once the system schematic is developed, it can be simulated in the SPW environment to check for correctness.

Figures 20, 21, and 22 show the transmitted signal, the received down-converted signal, and the FFT of the deramped signal. Note that it indicates a major response at a single frequency. The FFT is normalized to the sampling frequency (70 GHZ). When multiplied by 70 GHZ, this gives an actual peak frequency of 10.1465 MHz.

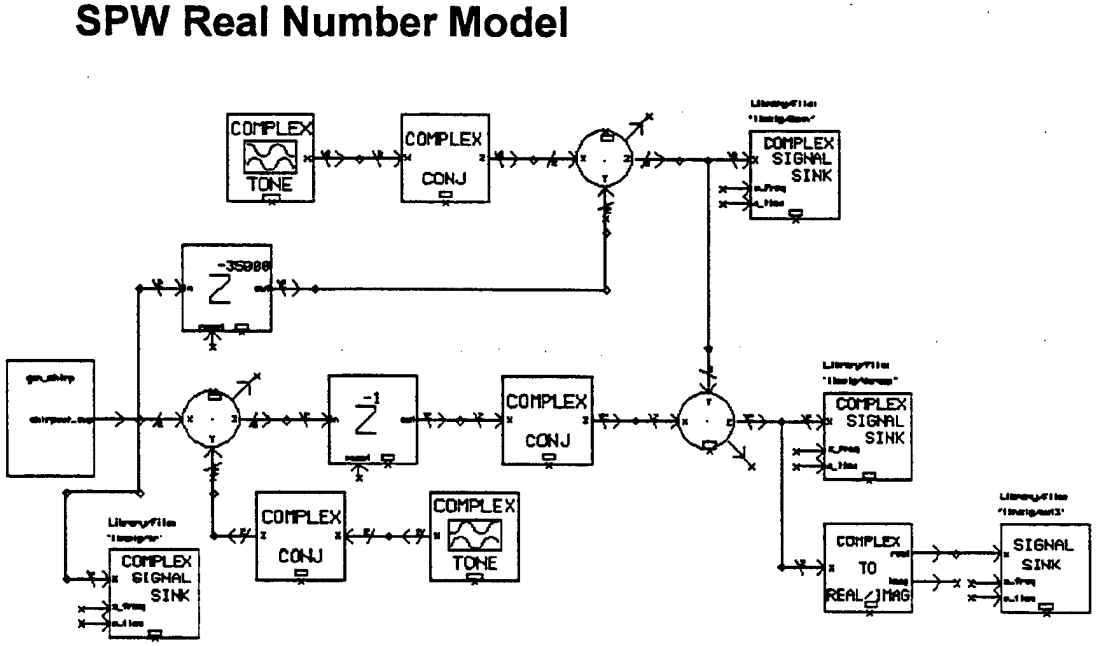


Figure 19. SPW Real Number Model

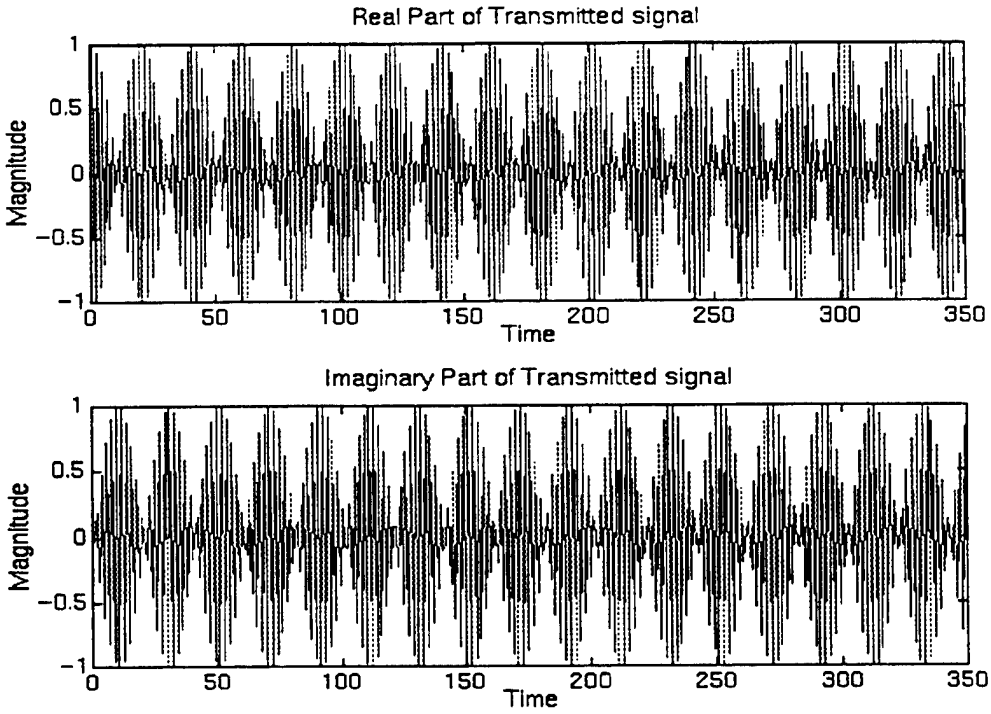


Figure 20. Transmitted Signal

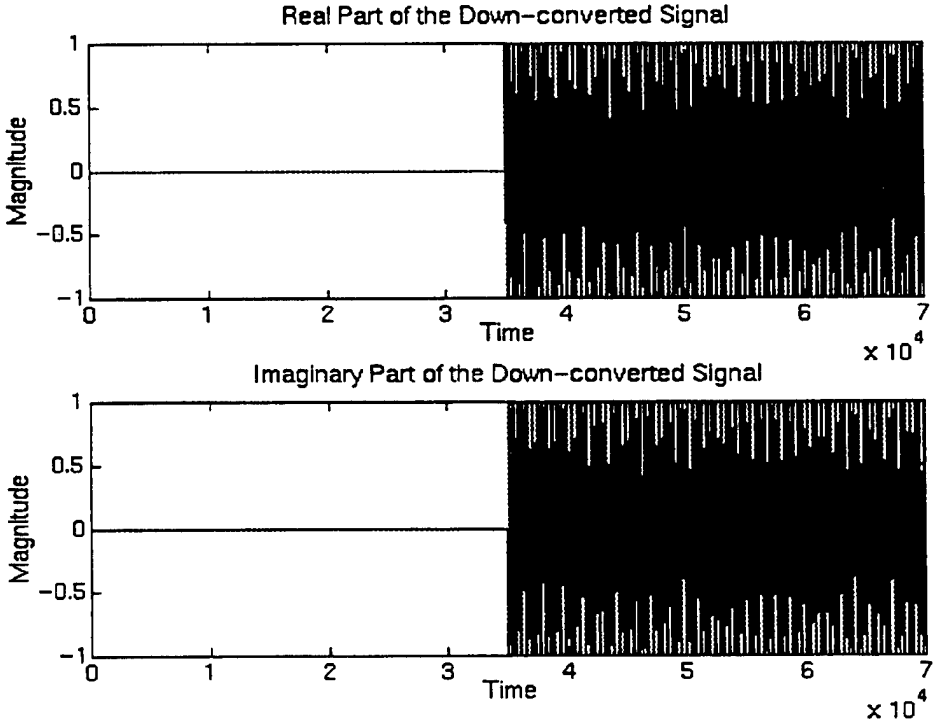


Figure 21. Downconverted Received Signal

Frequency analysis of the output data

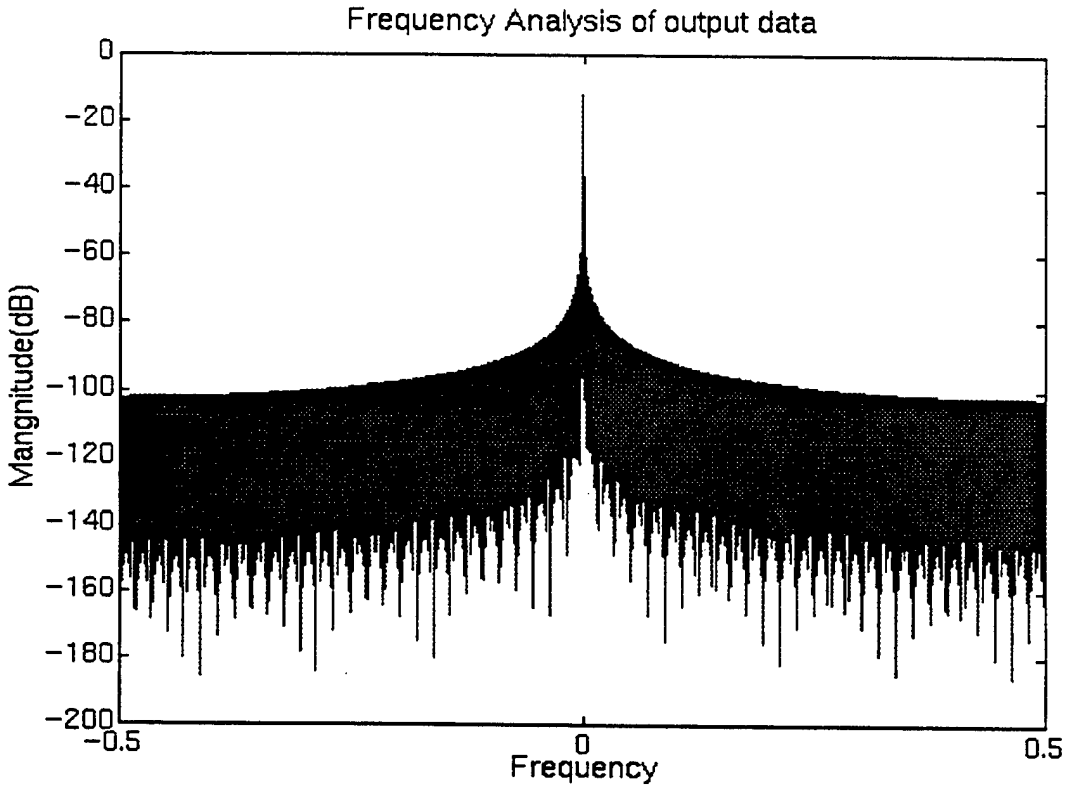


Figure 22. FFT of Deramped Signal

SPW has a code generation interface which can be used to generate C and VHDL. However, the VHDL interface is limited to fixed point models, probably because these models can then be further processed by automatic synthesis tools to yield hardware implementations. In our case, though, we were interested in real number models. To solve this problem, we developed our own SPW to VHDL conversion tool which could convert SPW models to VHDL real number models.

Figure 23 shows the system. SPW inputs to the system are the files underlying the block diagram, such as the one shown in Figure 19, and a file of model parameters. The other inputs to the system are VHDL real number primitives corresponding to DSP primitives that SPW uses in its schematics. We developed this VHDL primitive library. Thus, we achieved the capability to develop VHDL real number models from SPW.

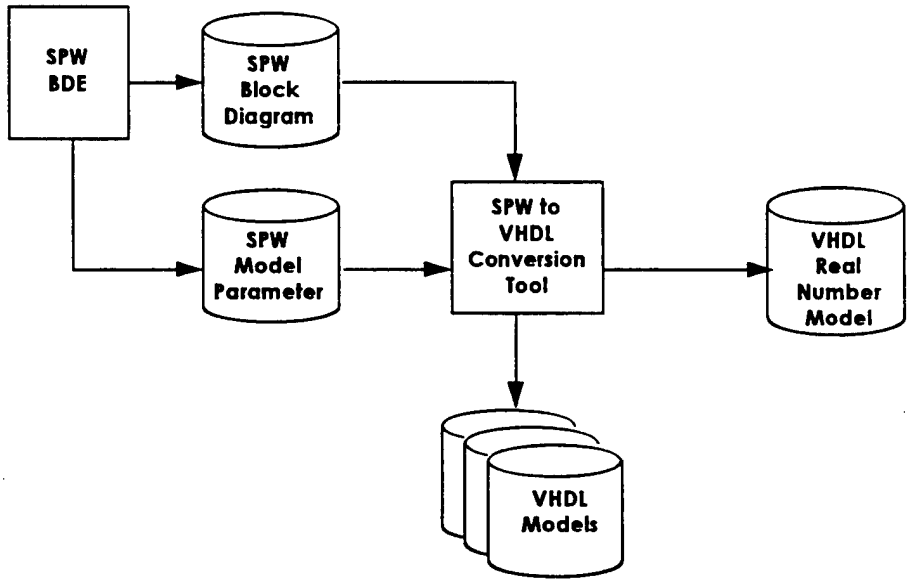


Figure 23. Real Number Model SPW To VHDL Interface

The data flow nature of the SAR also made activity charts an applicable modeling technique. Figure 24 is an activity chart model for SAR developed on Ilogix Express VHDL. The primitives used in this model are similar to the SPW primitives.

Ilogix Express VHDL Activity Chart

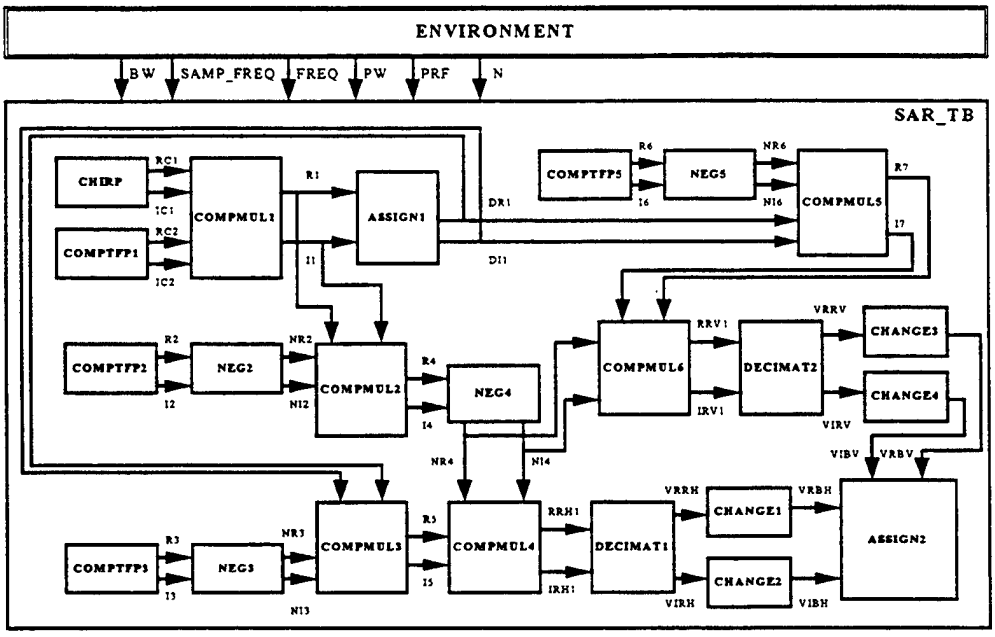


Figure 24. Activity Chart Model of SAR

2.2.5.5 Library Based Model Development

In this approach to model development for VHDL test benches, the test bench architecture is constructed as a structural architecture to form the model [GowP95]. The model instantiates primitives from a primitive library. The structural architecture is built using a commercial schematic capture tool. This approach promotes reuse of test bench structure and library primitives, and thus is important to the RASSP model year concept [GraG94, FraG95].

In our work, we used the Synopsys Graphical Environment(SGE) [Sy92] as our schematic capture tool. SGE contains a Symbol Editor which can be employed to create library primitives, a Schematic Editor to interconnect the primitives, and a VHDL interface to generate the VHDL description of the structural model that is created.

Our work with the behavioral modeling tools resulted in libraries for SAR and IRST test benches which were refined over time in the library structure. The libraries we developed contained the following components:

SAR Low-level components - chirp, complex tone, complex multiply, delay, complex conjugate, decimate, and type conversion.

High-level components - Genchirp, delay, down converter, deramper, merge, and noise.

IRST - target, clutter, sensor, clock

The function of the primitives listed above is mostly evident based on previous discussion or common DSP terminology, but two require explanation. The SAR merge component merges multiple single point target returns into one signal. The noise primitive injects noise into the radar signal. In IRST only four primitive types are listed, but there are many versions of these. Figures 25 and 26 show the structural test benches constructed for IRST and SAR using SGE and the primitive library.

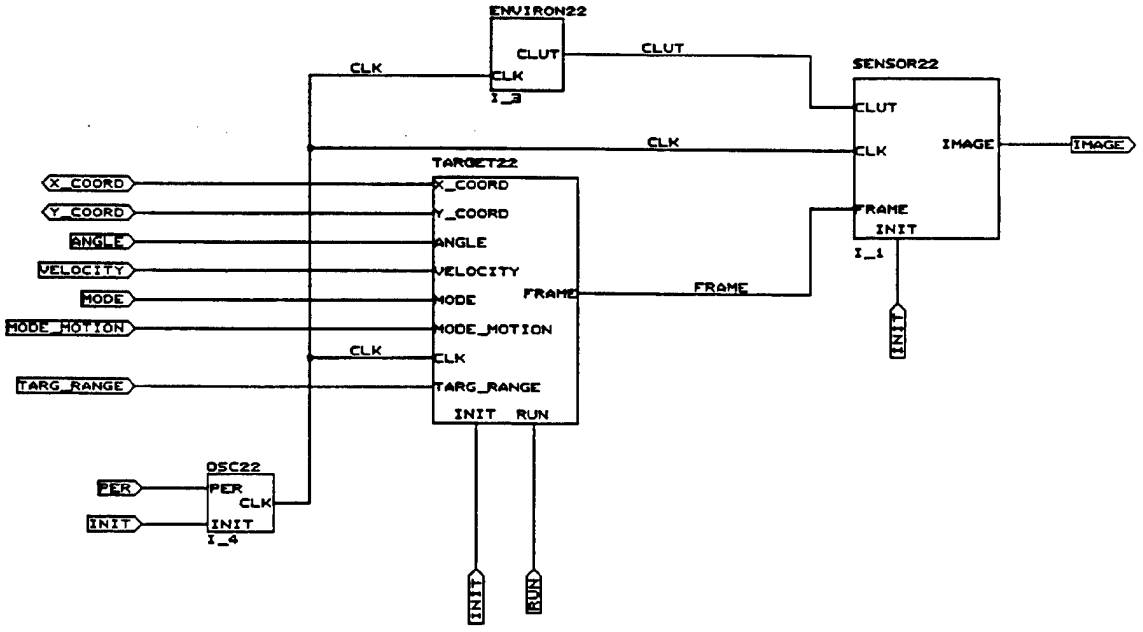


Figure 25. Structural Test Bench for IRST

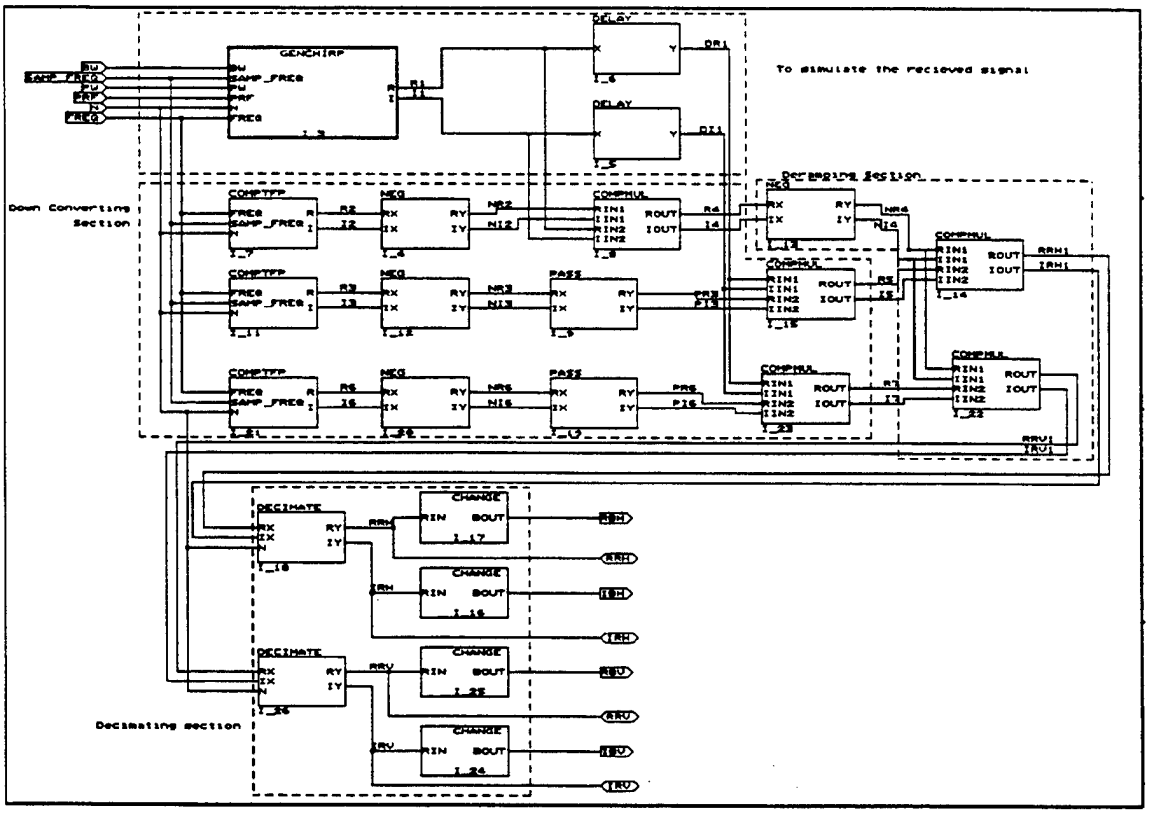


Figure 26. SAR Structural Test Bench

2.2.5.6 Model Simulation Efficiency Studies

As models were refined in the library setting, it was important to assess the efficiency of various modeling approaches. An important question in this area is the relative efficiency of signal based models (built structurally) to variable based models using procedures. In the DSP application the data flow is unidirectional; thus it is straightforward to develop a procedure based equivalent of an entity based model. In our study, the entity based models were created via the SGE approach, the equivalent procedure based models were done by hand.

Table 3 shows the simulation results . In the data the left column is the **VHDL simulation time**. The right two columns are **wall clock time** as measured by operating system probes. Note that procedure based models using variables are 10% to 20% faster than entity based models using signals. Other work done at

Virginia Tech [WicJ96] and other literature results agree with this [PauB92, BalA94]. Because of this, a schematic capture tool for procedure based models in which the procedure corresponds to a primitive would be useful.

Table 3. Simulation Results

Simulation Time in NS	Entity based testbench model using signals	Procedure based testbench model using variables
10 NS	5158886 US	4427989 US
14 NS	5881279 US	5183508 US
16 NS	7513527 US	6047736 US
20 NS	8130987 US	6541481 US

2.2.5.7 Domain Specific Environment Modeling Software

As part of our environmental modeling for RASSP test benches, we also employed domain specific software to develop model input data. It is very important to use this approach because the writers of this software typically have a better understanding of the application domain than the VHDL modeler would. One (manageable) problem with this approach though is that format conversion is typically required before the data can be used in the model. Also, these programs sometimes produce very large data files so efficient techniques for reading these files must be employed. Finally, the execution of these programs is frequently slow and so they may have to be run in an off line mode and the data read by the test bench later.

For IRST we originally used MATLAB for generating two-dimensional arrays representing simple targets and simple background clutter. Later we developed a more realistic approach to modeling IRST sensor inputs. For target signatures we used a tool called IRTOOL [Are95], which was developed by Arete' for the Navy. It produces realistic IR signature of cruise missiles. For background, we used the IRAMP data base [ONR94]. This data base of clutter files maintained for NRL which contains ocean scenes consisting of sky, sea, and clouds. To produce an IRST input, we just combine the IRTOOL missile signature with an IRAMP background picture. Figure 27 shows an example of this.

An example of continuous frames generated by the testbench with artificial target(includes target motion, platform motion and sensor noise)

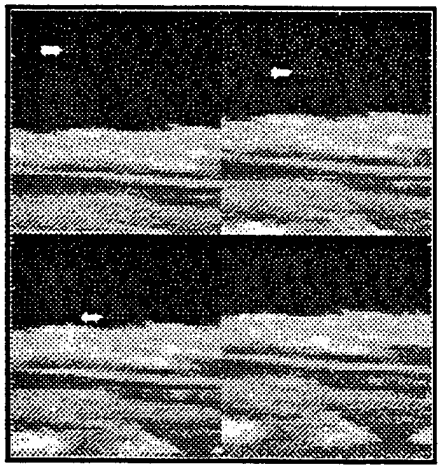


Figure 27. IRST Input Frames

For SAR environmental modeling, we used real data files of radar data from MIT as data files in an off-line test bench generation mode. An example of this will be shown below. We also employed, to a limited extent, a program known as xpatch [Dem93], which was developed by DOD to produce individual radar returns from and radar cross sections of various objects. However, our best success was with the use of SPW to model returns of single and multiple point targets. Figure 28 shows a radar return from two point targets where the two targets have been superimposed. This figure gives the time response of the returns. Below we will show the FFT of a similar case. Using SPW gave us the most control over return characteristics.

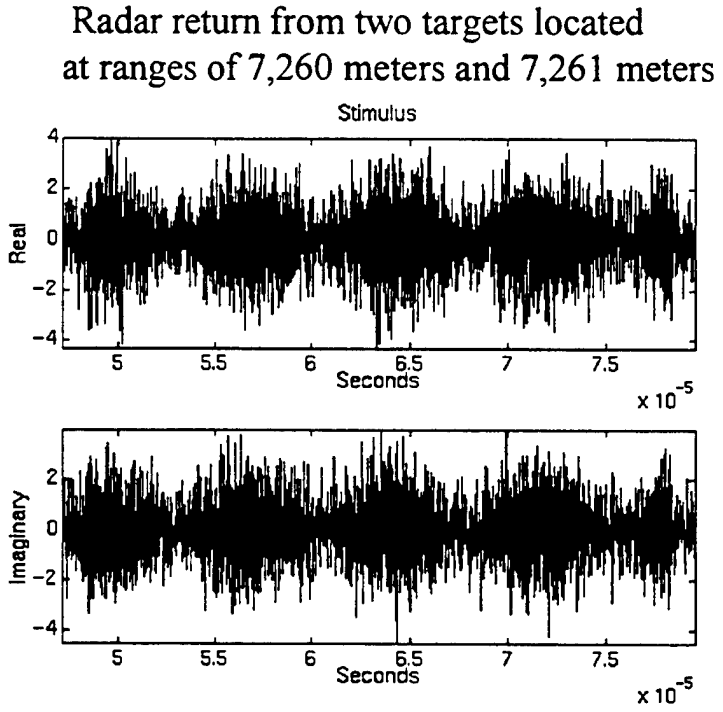


Figure 28. SAR Radar Return From Two Point Targets

2.2.6 Linkage to System Requirements

In our approach to test bench generation, our linkage to system requirements has the following characteristics[ArmJ95, ArmJ96] :

1. A **specification repository** holds primary specific requirements.
2. A math model, known as a **requirements interface**, receives primary requirements and generates derived requirements.
3. These derived requirements and some of the original primary specific requirements specify the values of test bench code generics.

2.2.6.1 Specification Repository

The specification repository is a high-level system diagram where blocks in the schematic correspond to real system components. It is intended for use by systems engineers who are not VHDL literate, thus underlying VHDL is hidden. Our system diagrams are developed using a schematic capture tool, Synopsys SGE. With this approach, specification parameters are **symbol attributes.** A parser extracts the values of the attributes (specification parameters) and feeds them forward to the test bench system. Thus, if a specification value is changed, its new value is automatically fed forward to the test bench. Figure 29 shows how a specification parameter is altered in the IRST system schematic. The systems engineer clicks on the block of the system component for which the specification values are to be changed. A window then pops up where specification values can be edited. In this example the frame time (the time between adjacent frames) is being edited. Figure 30 shows the system schematic for the SAR system.

System Level Schematic Diagram Of IRST System Showing Specification Modification.

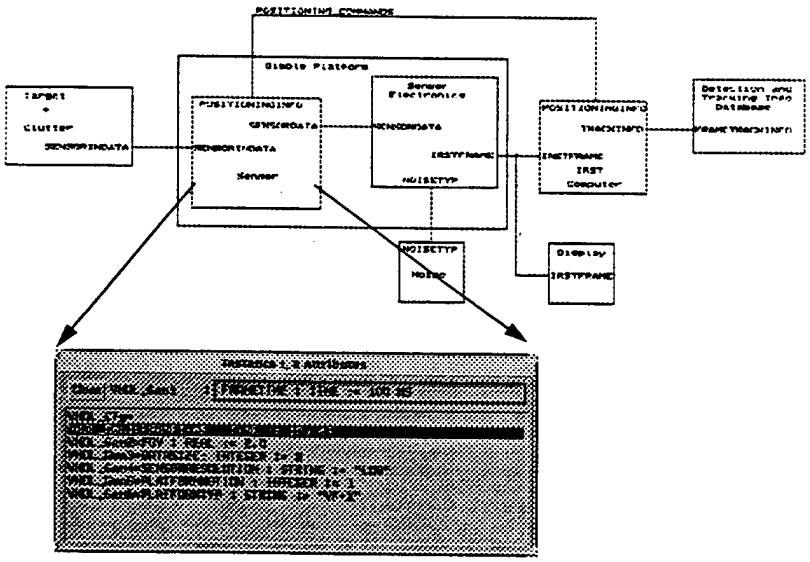


Figure 29. Changing An IRST System Parameter

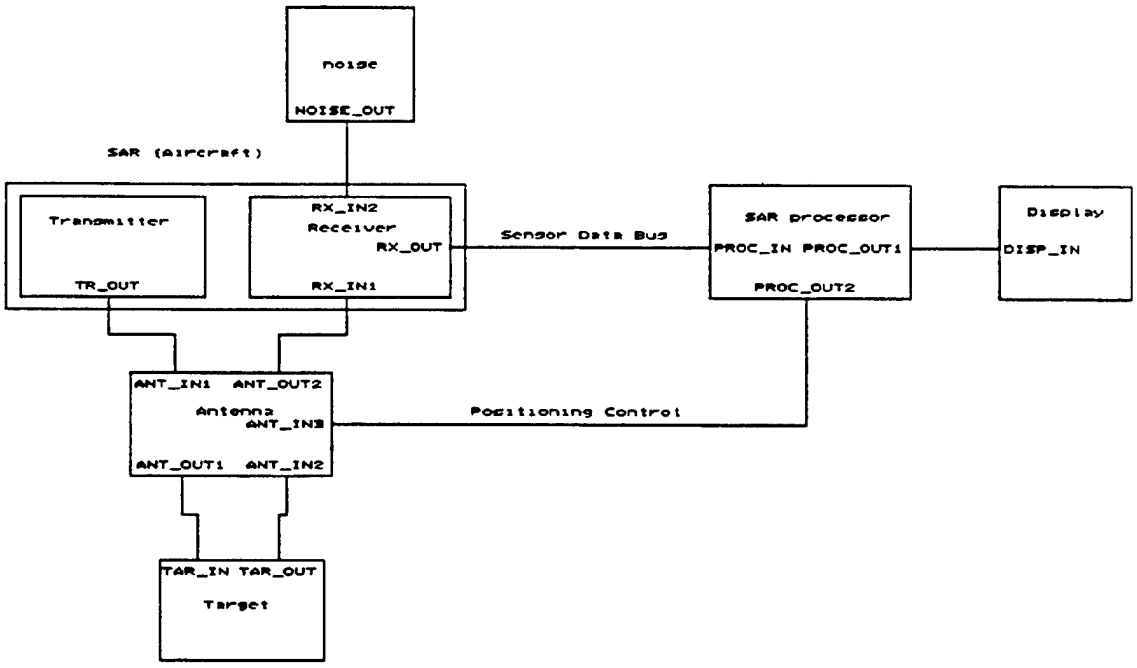


Figure 30. SAR System Schematic Diagram

2.2.6.2 Requirements Interfaces

As indicated above, a requirements interface is a math model that receives primary specific requirements from the specification repository and generates derived requirements. These derived requirements and some of the original primary specific requirements are the values of test bench generics. Figures 31, 32, and 33 show the requirements interface for IRST and its associated math model. The purpose of this model is to translate sensor properties and platform velocity into platform displacement and clutter motion measured in terms of pixels.

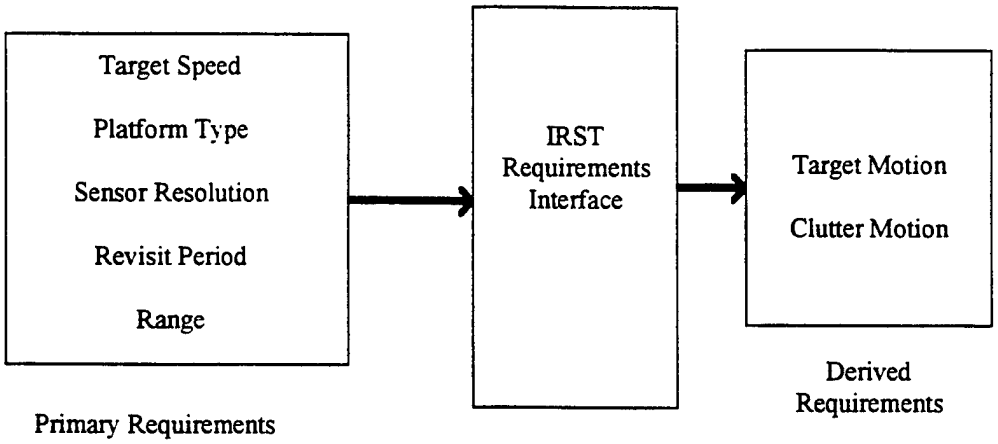
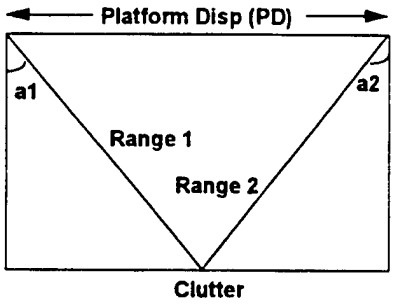


Figure 31. Requirements Interface for IRST

- List of equations
  - Sensor Res factor = 100/1000000 if Sensor Res = Hi
  - Sensor Res factor = 250/1000000 if Sensor Res = Low
  - Platform Velocity = 448 m/s if Platform type = VF-X
  - Platform Velocity = 256 m/s if Platform type = VF
  - Platform Velocity = 192 m/s if Platform type = VP
  - Clutter Range = 1609.344 * Range(200 miles)
  - Platform Disp = Platform Velocity * Revisit Period(1 sec)
  - Clutter Motion = 2 *asin (Platform Disp/ (2 * Clutter Range)) / Sensor Res factor

Figure 32. Math Model for IRST Requirements Interface



Total Angular Disp = a1 + a2

Assume a1 = a2, Range1 = Range2

Sin a1 = (1/2 * PD) / Range1

a1 + a2 = 2 * asin(PD/2 * Range1)

PAFOV = Pixel angular field of view

Pixel Disp = (a1 + a2) / PAFOV

Target Range = 1609.344 * Range(200 miles)

Target Velocity = Mach to m/s(320) * Target speed

Target Disp = Target Velocity * Revisit Period(1 sec)

Target Motion = 2 * asin (Target Disp / (2 * Target Range)) / Sensor Res factor

Figure 33. Math Model for the Requirements Interface (Cont'd)

Figures 34 and 35 show the requirements interface and the math model for SAR. This math model is considerably simpler than the IRST math model. A case could be made for putting these calculations in the VHDL code. However, having them external makes them controllable by systems engineers. Also, putting them in the VHDL models may limit the generality of the modeling primitives. We did not explore this issue extensively, though.

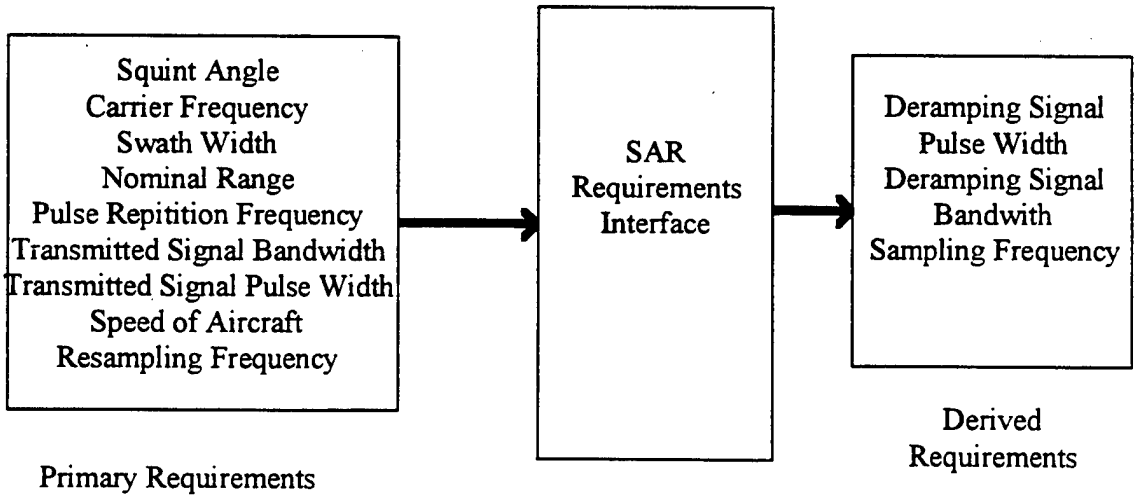


Figure 34. Requirements Interface for SAR

•Pulse width of the signal used to do de-ramping
Pulse width of transmitted signal +(swath width / speed of light)
=30 μs+2*(375m / 3*108(m/s)) = 32.5 μs.

•Bandwidth of the signal used to do de-ramping
Rate of change of frequency * pulse width of the signal
= (600 MHz /30 μs) * 32.5 μs = 650 MHz

•Sampling frequency
Sampling frequency > 2 * carrier frequency
= 70GHz > 2*33.56GHz.

Figure 35. Math Model for SAR Requirements Interface

2.2.7 Test Plan

A test plan is a document that organizes system requirements in terms of how the requirements will be tested. It divides requirements into test groups where one particular system requirement is left unspecified and other system requirements receive fixed values. A set of tests is then allocated to each group.

The Test Plan Interface that we developed for the Test Bench Generator uses the following approach:

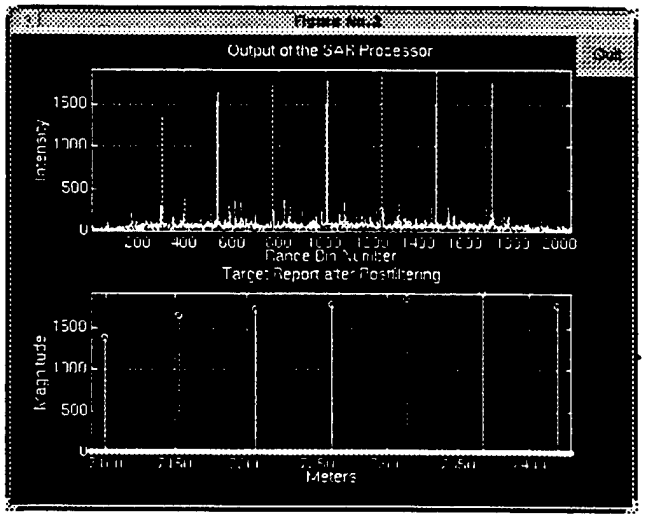
1. For each application (SAR or IRST), the test bench is an unbound structural architecture.
2. A VHDL configuration body specifies which library models to use and assigns values to generics.
3. A test group corresponds to a partially specified configuration body.
4. Each individual test corresponds to a fully specified configuration body.
5. The library models are those described in 2.2.5.5.

For the RASSP test program, demonstration test plans were developed for both SAR and IRST.

For SAR the test plan consisted of four test groups with the following goals:

1. Evaluate the range of a single point target.
2. Evaluate the range of multiple point targets.
3. Evaluate the discrimination of two point targets.
4. Evaluate SAR Algorithm noise sensitivity.

Figure 36 shows the results of test Group II test in which seven targets were detected. The figure shows the FFT of the SAR algorithm output. Each target corresponds to a discrete frequency. Figure 37 shows the test results for a test from Group III. Here, two targets 0.25 meters apart can still be discriminated. Figure 38 shows the results of a noise test from Group IV where the correct target at 7260 meters was detected, but because of the noise, two "ghost" targets were also detected.



- 7 targets applied
- All targets detected
- MSE 0.12 meters

Figure 36. A Test Case of Test Group II

- 2 targets applied (7,260 m and 7,260.25 m)
- Spacing = 0.25 meters
- Can be discriminated

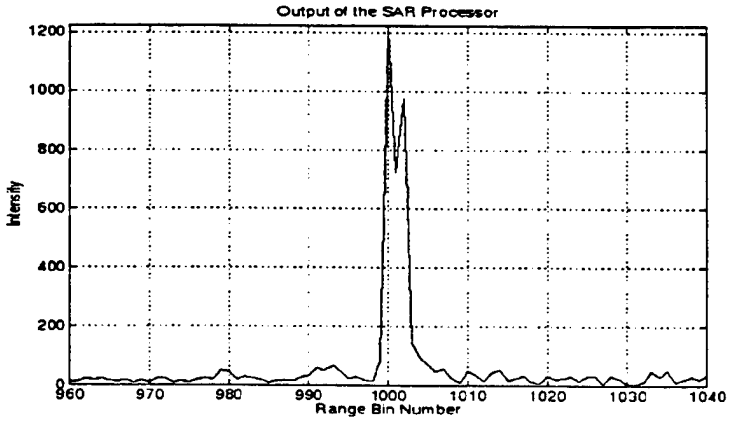


Figure 37. A Test Case of Test Group III

A Test Case of Test Group IV

- One target applied (7,260 meters)
- Gaussian noise standard deviation = 6.6
- Three targets detected (7200.13, 7215.68, 7260.05 meters)

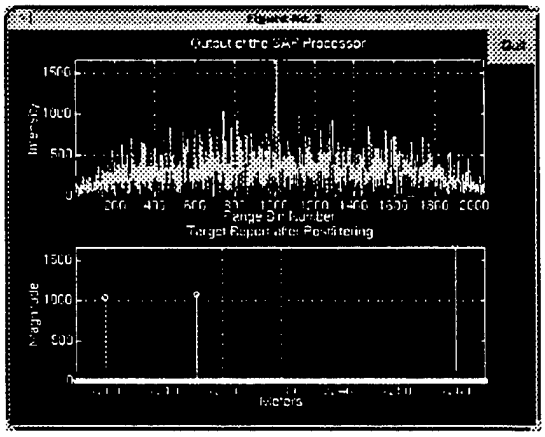


Figure 38. A Test Case of Test Group IV

For IRST the test groups had the following goals [Kots96]:

1. Target displacement detection
2. Sensor frame displacement detection
3. Noise sensitivity determination
4. General test group

The general test group allows the user control over all test parameters. The test combines target motion, platform motion, and sensor noise.

2.2.7.1 Iterative Test Mode

This mode involves repeated execution of the test bench generation and execution software. Its purpose is to use repeated execution to determine the limiting value of a system parameter such as noise sensitivity.

Figure 39 summarizes the application of the iterative mode to SAR. In the right-hand column are the test strategies. "Even division without end points" means that the single targets tested are equally spaced within a major interval and end points are excluded. "Constant increment" means that the number of multiple targets is incremented from one test bench iteration to the next. " Double increment and binary search" means that the test parameter is incremented until an upper and lower bound is found on the parameter. After which a binary search is used to determine where the value is within the two limits.

Test Group	Adjustable Parameter	Test Strategies
Range of single target	Target range	Even division without endpoints
Ranges of multiple targets	Number of targets	Constant increment
Discrimination of two targets	Distance between two targets	Double increment & binary search
Sensitivity to noise	Noise standard deviation	Double increment & binary search

Figure 39 SAR Test Groups of Iterative Test Mode

Figure 40 shows the results of the iterative mode for SAR Test Group II where the number of multiple targets is incremented from one to seven. MSE is the mean squared error in the range of the detected targets vs. the inserted targets. Figure 41 shows the results for SAR Test Group III where eight iterations were used to determine that two targets can be detected as long as they are 0.23 meters apart. Figure 42 gives the results of an iterative test sequence that determined that the maximum tolerable noise standard deviation is 6.1.

Test No.	1	2	3	4	5	6	7
No. of Applied Targets	1	2	3	4	5	6	7
No. of Detected Targets	1	2	3	4	5	6	7
MSE (meters)	0.05	0.15	0.09	0.14	0.13	0.16	0.09
Pass/Fail	pass	pass	pass	pass	pass	pass	pass

Figure 40. Results of Iterative Mode for SAR Test Group II

- Initial distance = 0.05 meters
- Precision = 0.01 meters
- Finished in 8 iterations
- Discrimination capability = 0.23 meters

Figure 41. Results of Iterative Mode for SAR Test Group III

- Adjustable parameter: noise standard deviation
- Test strategies: double increment & binary search
- Initial noise standard deviation = 1 (normalized to the amplitude of chirp signal)
- Precision = 0.1
- Finished in 10 iterations
- Maximum tolerable noise standard deviation = 6.1

Figure 42. SAR Test Group IV Evaluation of Noise Sensitivity

The results of the iterative mode tests for IRST are shown in Figure 43. In these tests an increment strategy was used to determine maximum value for target speed, platform speed, and noise level.

Test Group	Parameter	Increment	Limiting Value
Target displacement detection	Target Speed	25 m/s	450 m/s
Sensor frame displacement detection	Platform Speed	15 m/s	225 m/s
Noise sensitivity	Noise Level	10	110

Figure 43. Results of Iterative Mode for IRST

In a follow-on contract sponsored by the U. S. Air Force, we will combine the test plan with goal trees to develop a high-level approach to system testing. A goal tree will represent a test plan. The goal tree is given a grand goal such as "determine the system noise sensitivity." The grand goal is decomposed into

subgoals until primitive goals are reached. Primitive goals define a test group that would be applied to reach the grand goal.

2.2.8 System Integration

Figure 44 is a modification of the basic approach shown in Figure 8, as it shows a high level test bench generation system with test plan interface. Now default specification requirements are received from the Specification Repository. The Test Plan Interface can override these default requirements or modify them based on the test group to be employed. The Test Plan Interface produces the final specific primary requirements and forwards them to the Requirements Interface which employs a math model to produce derived requirements which are the generic values. The Test Plan Interface also produces model selection information which selects test bench primitive models to bind to the test bench structural architecture.

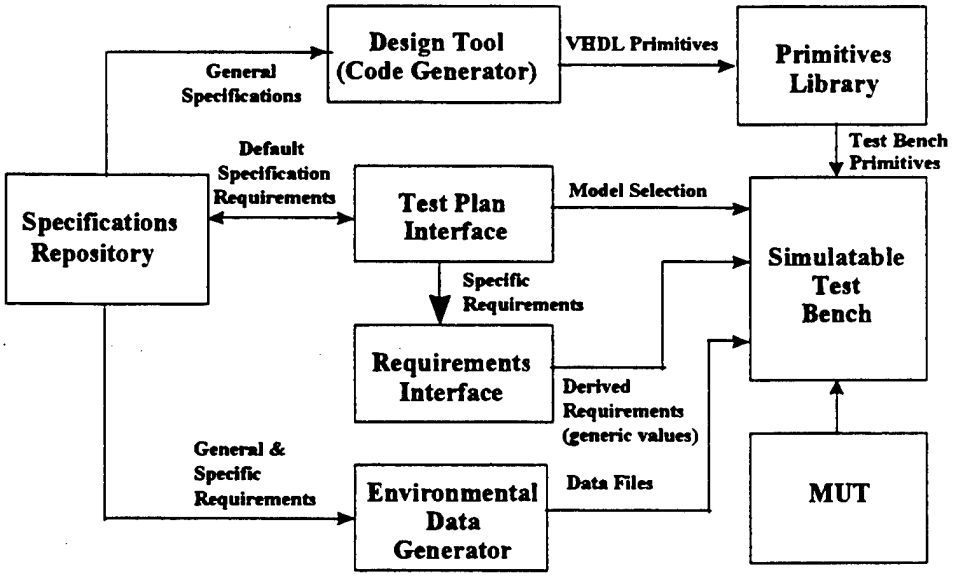


Figure 44. A High-Level Test Bench Generation System with Test Plan Interface

The other information flow paths are the same as in the basic system. General specification information is used by the Design Tool to develop VHDL primitives which are refined in the library structure. As time passes, the Design Tool is used infrequently and test benches are just developed structurally using the

primitive library. Both general and specific requirements are still used by the Environmental Data Generator to produce data files for test benches that use file I/O.

2.2.8.1 Systems Integration Work

Existing design and application area tools can be used to develop pieces of a test bench. However, two software systems are required to integrate the pieces. Figure 45 shows the Test Bench Generation System which produces the test bench. The top part of the diagram has been explained: Specification Repository, Design Tool, Requirements Interface, and Environmental Data Generator. Now three libraries are shown: Test Data Library, VHDL System Database, and TB VHDL Components. The Test Bench Generation User Interface (TBGUI) interacts with the user to select test plans, models, generic information and data files that make up a test bench. At each step, one can accept or override default values. TBGUI also allows one to edit the specification information in the Repository. The outputs of the Test Bench Generation System are: 1) the complete executable VHDL test bench, 2) test data files to be read by the test bench during execution, and 3) a file of simulation control information. This file is used to control the simulation and is peculiar to the VHDL simulator being used [Syn95,Van92] .

Figure 46 shows the VHDL Simulation Controller. This system uses the VHDL test bench model, test data files, and simulation control files to perform the simulation. The Test Bench Execution User Interface allows the user to select the modes of execution and output display.

Test Bench Generation System

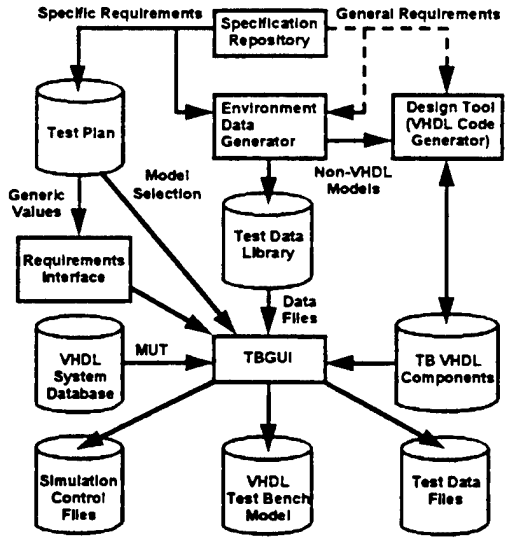


Figure 45. Test Bench Generation System

The VHDL Test Simulation Controller

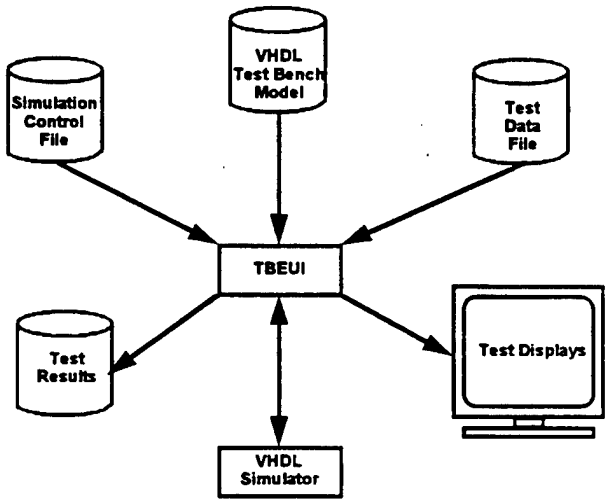


Figure 46. The VHDL Test Simulation Controller

2.2.8.2 Demonstration System

For our laboratory demonstrations system, we combined the Test Bench Generation System and the VHDL Test Simulation System into one system with one menu driven user interface. The software is written in C using X-View layer libraries of the X-11 protocol. The interface prompts the user with a list of possible selections or prompts the user to key in certain values in a certain range. Figure 47 shows the menu structure for IRST (SAR is similar). The user first selects file I/O or a code generated (on-line). For the file I/O case one next selects the results of a particular test plan that were derived previously off line. Within the results of a particular test plan, the results of a particular test case are selected which results in the selection of data files to be read by the test bench. Finally one chooses either on-line or post processing of simulation output. With on-line processing, X windows show results as they are generated. With post processing, MATLAB is used to process the results after simulation. Post processing is generally faster. After this selection, simulation begins and results are displayed. Figures 48, 49, and 50 show input frames read from a file by a test bench, X window online displays, and final test displays which show the comparator output. Figure 51 shows the result of a SAR file I/O simulation. The three-dimensional figure depicts computed range and azimuth information in a received SAR signal. The input file was from MIT.

User Interface Menu Structure for IRST

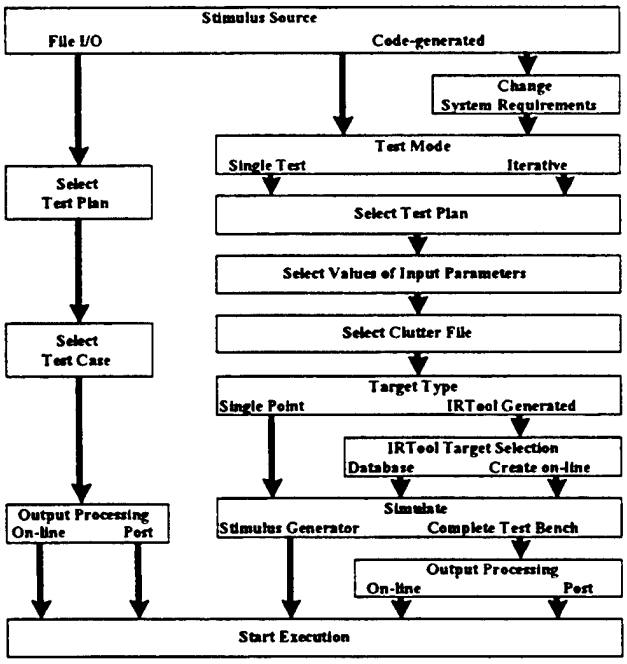


Figure 47. User Interface Menu Structure for IRST

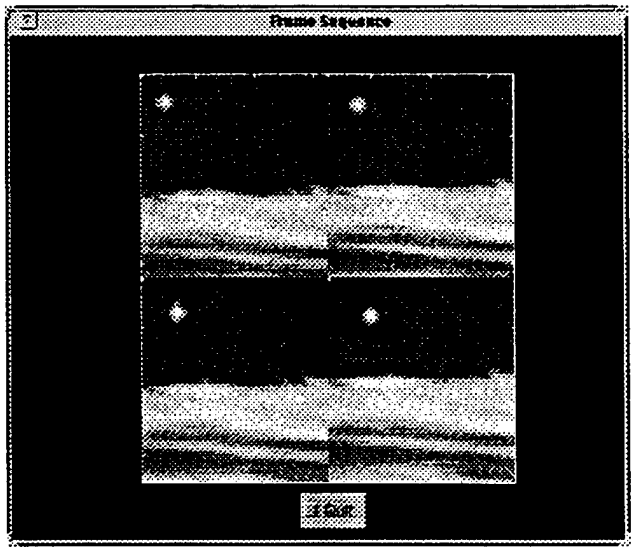


Figure 48. Test Displays: Test Vectors Used in the File I/O Test Case

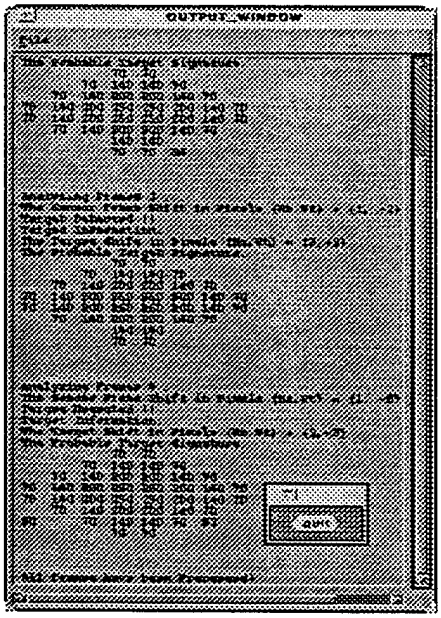


Figure 49. Test Displays: MUT Outputs for the File I/O Test Case

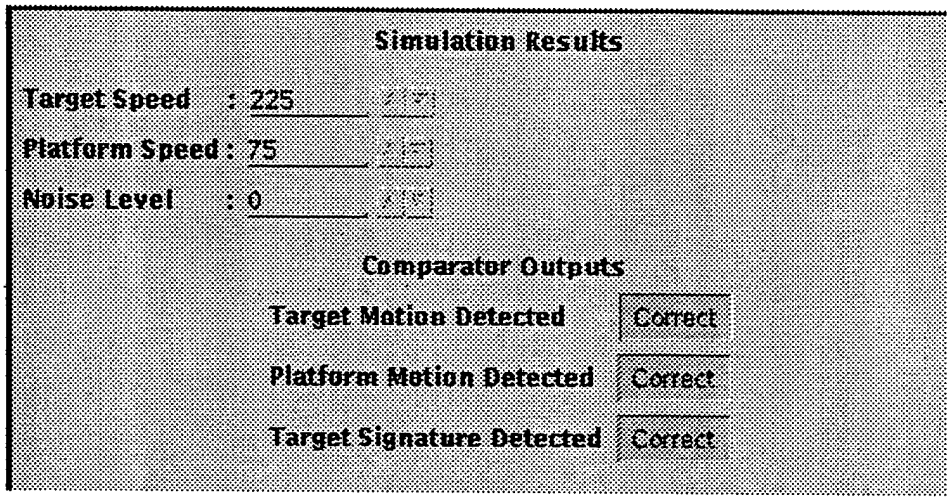


Figure 50. Test Displays: Comparator Outputs for the File I/O Test Case

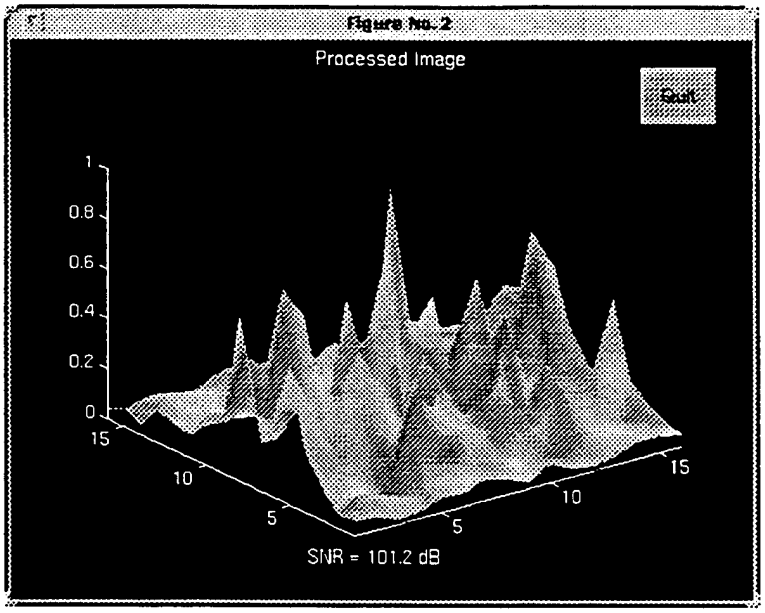


Figure 51. SAR Simulation Result Using File I/O Mode

The right side of the menu structure is for code-generated inputs (online) test benches. In this mode, one first has the option of changing information in the Specification Repository. Next, one selects the test plan and values of free parameters in that test plan. The next several choices relate specifically to IRST. One chooses a clutter file and whether to use a point target or an IRTOOL signature generated one. For the IRTOOL choice, the target can be read from a data base (fast) or generated online (slow). The final two selections are shared by the SAR system. One selects whether to simulate the test bench only or the test bench driving the MUT. Finally, the method of output processing is chosen.

Figures 52, 53, and 54 show setting up and results of a SAR multiple target simulation. In Figure 52, the user selects the multiple target test plan, the number of targets, and their ranges in meters. The rest of the specific requirements are provided by the Specification Repository. Figure 53 shows the FFT of the SAR algorithm output as displayed by MATLAB. Figure 54 gives X window target reports which list detailed numeric data on each of the targets detected and the mean squared error in their combined detection.

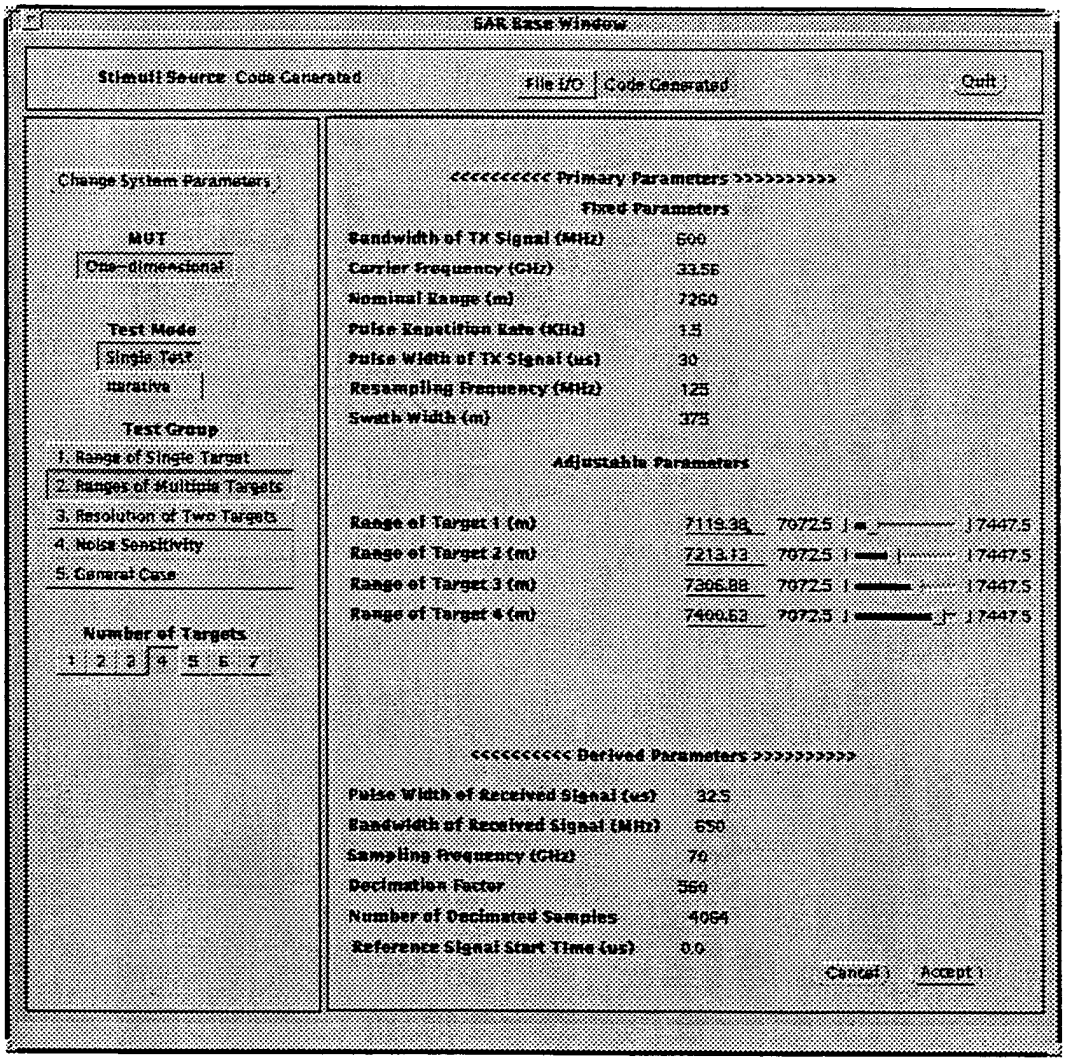


Figure 52. SAR TBGUI Base Window Showing Code Generated Mode and Parameter Entering

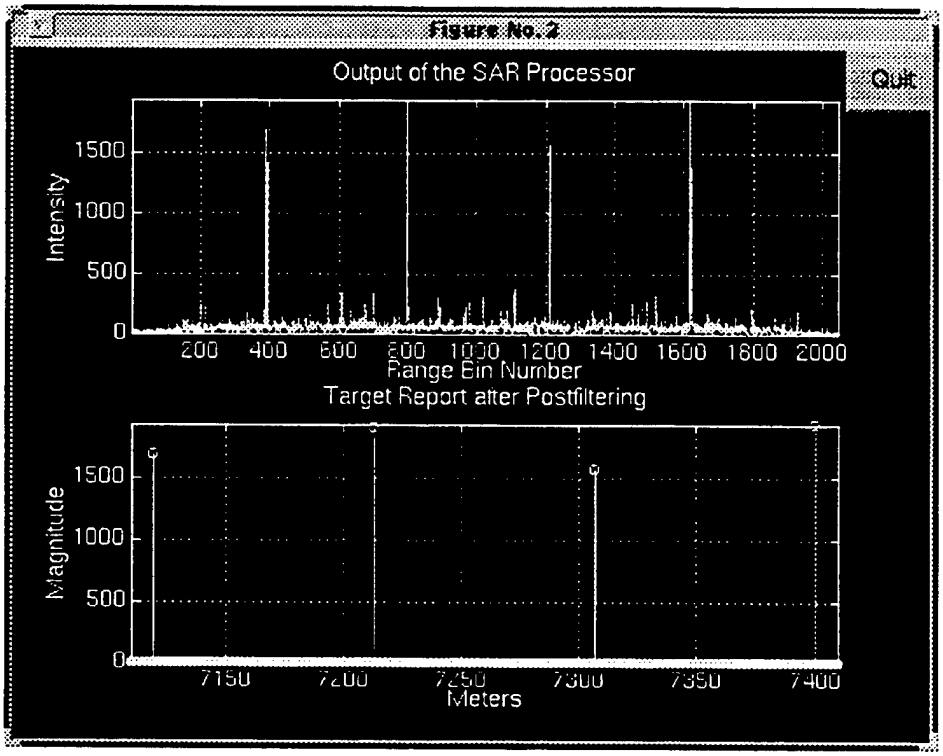


Figure 53. Code Generated Mode of Test Group II: Range of Multiple Targets

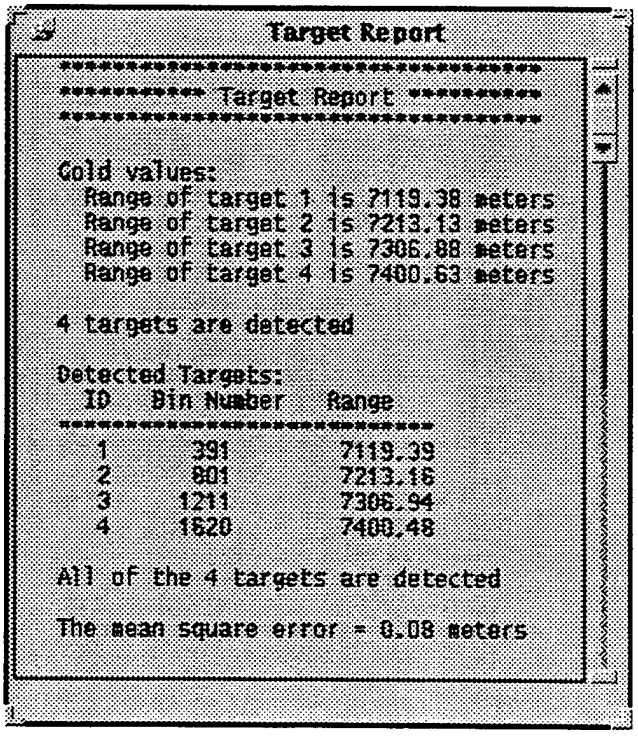


Figure 54. Target Report

The Test Bench Generator was written for execution on Sun workstations. To execute the program, Synopsys VHDL Analyzer and Simulator, and Synopsys Graphics Environment and MATLAB are required.

2.2.9 Conclusions

In our RASSP sponsored work we have shown that effective test bench generation requires:

1. High level graphics design tools to develop initial test bench code quickly.
2. A test bench component library to construct test benches structurally.
3. Accurate environmental modeling using application domain specific software.
4. Automatic linkage to the system specification.
5. A test plan interface to configure the test bench structural model.

And while commercial software can be used for generation of test bench pieces, system software having graphic user interfaces is needed to integrate the pieces into a working system.

3. Accomplishments

3.1 Tool Construction

3.1.1 Algorithm Partitioning Tool

Development of the Algorithm Partitioning Tool consisted of three efforts conducted recursively:

- Selection of tools and libraries,
- Development of target models, and
- Development of interfaces.

Initially, simple models were constructed using candidate toolsets to determine toolset and library suitability and to provide specifications for interface inputs and outputs. This was followed by development of models of increasing complexity at each stage, algorithm, hardware, spreadsheet, and performance models, and determining if sufficient data was available from the predecessor stage to support the following stage. When not, "give and take" modifications were made to the models. In order to keep the interfaces simple and understandable, we decided at the start of the contract to make all interface inputs and outputs ASCII files.

3.1.1.1 Selection of Tools and Available Libraries

For VHDL modeling, it was decided at the outset to use the Honeywell developed PML library and the Synopsis compiler and simulator for both the APT and the test bench generator. However, the PML models were developed by Honeywell using the Vantage VHDL simulator (now Viewlogic Optium VHDL simulator). Even with extensive support from Honeywell personnel, we were unable to get even simple models to simulate using the Synopsis VHDL simulator. At the midpoint of our contract, we decided to adopt the Vantage VHDL simulator for APT VHDL simulation, and retain the Synopsis VHDL simulator for the test bench work. Since the VHDL performance model development was behind schedule due to

unanticipated problems with the VHDL simulator and to debugging problems with the PML models. we developed an extraction interface for schematics drawn with Network II.5 to allow work on the tool interfaces to progress in parallel with the PML model development. As PML models became available, we were able to use the Network II.5 experience to accelerate VHDL interface development.

Team members attended early meetings at both RASSP prime contractors where it was determined that ComDisco's SPW and BONeS and U. C. Berkley's Ptolemy Synchronous DataFlow and Discrete Event Domains were prime candidates for functional and non-VHDL performance simulation respectively. We visited both U. C. Berkley and ComDisco and for the first 6 months of the contract, we did initial modeling with Ptolemy, SPW, and BONeS. At that time, our preference was to use Ptolemy. However, when we received the Lincoln Laboratories SAR specification and started modeling it, we found that the package was not robust enough (in Jan/Feb 1994) and refocused on SPW/BONeS. It was later found, and confirmed by Cadence Alta Group, that there is no way to generate a BONeS model other than through their graphical interface. We then turned to Network II.5 as our performance modeling tool because it was robust enough to support the SAR application and CACI was willing to work with us and their tool has an ASCII interface.

A survey of all available spreadsheets was conducted at the outset of the project. The 2020 spreadsheet from Access Technology was adopted as it is the only one that provides a facility for creating and modifying spreadsheets with externally produced ASCII files.

3.1.1.2 Development of Target Models

Our approach was to build source and target models and then develop interfaces to transition from source to target. Also, we started with small, easily verified applications and simple architectures and worked our way up to real applications and architectures. The simple models were derived from the real target algorithms.

We started with a textbook SAR and notional IRST algorithms with which we were familiar and an architecture consisting of arrays of TI TMS320C40s. These models were used initially to familiarize ourselves with candidate tools, Ptolemy SDF, SPW, BONeS, and PML. We also started constructing exploratory spreadsheets to determine meaningful forms for presentation to the user.

Upon receipt of the Lincoln Laboratories Benchmark I SAR specification, we focused on that algorithm. The first models were a small model of the range processing and corner turn for a single polarity pair with very small vectors so that we could verify the output of each block of the SPW model. This was followed by a single polarity pair model of the entire algorithm that treated the azimuth processing as a single subswath. We then incorporated the kernel set selection and subswath processing into the single polarity pair model. Finally, we modeled the entire algorithm as specified. All four functional models and BONeS performance models (on the C40 arrays) were completed along with a demonstration interface for generating a sizing spreadsheet for the small model by the first RASSP conference.

At the first RASSP conference, it was determined that the prime contractors were both using the Mercury Raceway with i860 or sharc as their architecture. We focused or hardware and performance modeling on that architecture.

By the second RASSP conference, we had developed four running VHDL models using the Honeywell PML library components along with some additional in-house components.

1. Single processor, single global bus system.
2. Four processor, single global bus system.
3. Four processor, multiple local bus system interconnected by a crossbar switch.

The software tasks implemented were parts of the range processing and corner turn algorithms from the MIT Benchmark I SAR processing algorithm. The results of experiments performed on these models is reported in Hormazd P. Commissiariat's thesis (Performance Modeling of Single Processor and Multi-Processor Computer Architectures) and in a paper at the second RASSP Conference (Developing Re-usable Performance Models for Rapid Evaluation of Computer Architectures Running DSP Algorithms, pp. 103-108).

The final model was a 16-processor Mercury Raceway architecture. Two algorithms were mapped onto this architecture: 1) the SAR range processing algorithm, and 2) the single polarity multi-swath SAR benchmark. This work was reported in Srilekha Vuppala's thesis (Methodology for VHDL Performance Model Construction and Validation).

With these models available, we were able to finish the VHDL interfaces for APT.

3.1.1.3 Development of Interfaces to Generate Target Models

A key issue that drove interface design was maintaining information about predecessor/successor relations between functions (i.e., dataflow). For the SPW to spreadsheet interface, we needed to derive predecessor/successor relationships between our higher-level functions and the SPW primitive functions. This was required in order to determine iteration rates for spreadsheet functions. Within the spreadsheet, macros had to be developed to derive new predecessor/successor relationships as functions were mapped onto different processors requiring the insertion of data transfer functions.

At the first RASSP Conference, we demonstrated spreadsheet command files and scripts to build a prototype spreadsheet from manually produced hardware and software description files. Work had commenced on extracting the data for the software description files from an SPW model. Specs had been developed for extraction of component characteristics and architecture data from VHDL models.

By the second RASSP Conference, the SPW to spreadsheet interface was maturing and the library was being improved and expanded. The command file library was being expanded in parallel. Very small VHDL models had been built but no VHDL to spreadsheet interface yet existed. The SPW to spreadsheet interface was developed in four steps.

1. First, using the small (range processing and corner turn) algorithm we produced an SPW to spreadsheet interface. We were unable to extract control data, essential for performance modeling, with the SPW Tool Interface Language.
2. We solved the control problem by collecting the necessary data from an SPW file (model_name.mseq) generated when the functional model is simulated or when it is used to generate a C program. With this interface, we were able to proceed to the full single polarity pair algorithm without subswaths.
3. When we progressed to the single polarity pair algorithm with subswaths, we found a need to re-optimize the loop structures. The model_name.mseq file is optimized for a single, not multiple, processors. We were able to add loop re-optimization to the interface and produce satisfactory spreadsheets for the full single polarity pair algorithm with subswaths.
4. Finally, it was found when we proceeded to the full benchmark algorithm, which is really three threads of the same algorithm, that it would be extremely difficult for the user to distinguish between threads. SPW has no way to differentiate between different instantiations of a function. This was solved by adding a parameter named thread to all of our library blocks. This completed our SPW to spreadsheet interface.

We decided to extract HW data from a Network II.5 schematic in order to start development of the hardware model to spreadsheet and spreadsheet to performance model interfaces. This work paralleled the SPW to spreadsheet interface efforts and in most cases drove the requirement for changes in that interface.

Development of the Network II.5 interfaces provided structure and insights that supported rapid development of interfaces with VHDL models as soon as VHDL target models were developed. In the last year of the contract, the VHDL interfaces were developed.

1. VHDL processor characteristic files to APT processor characteristic files interface. This interface was written using VTIP (VHDL Tool Integration Platform from CAD Languages, Inc.) to parse the processor characteristic files provided in the Honeywell PML library. Essentially, VTIP provides a set of callable C.
2. Language subroutines that allow searching any VHDL language file for specific constructs. Priya Balasubramanian wrote a C program called PCET (Processor Characteristic Extraction Tool) that uses the VTIP C routines to extract processor parameters needed by the APT sizing spreadsheet. These parameters are stored in the Processor Characteristic Library described in Section 2.1.3. This tool is documented in her thesis entitled Interfacing VHDL Performance Models to Algorithm Partitioning Tools.
3. VHDL structural model to APT Spreadsheet interface. Priya Balasubramanian also wrote a C language program called ACET (Architecture Characteristic Extraction Tool) using the subroutines in VTIP. This interface parses a VHDL structural model of the target architecture built using PML components in order to extract parameters from each component in the candidate architecture that are needed by the APT partitioning spreadsheet. This tool is also documented in her thesis.
4. VHDL structural model to APT spreadsheet interface. Priya Balasubramanian also wrote a C language program called CONET (CONection Extraction Tool) using subroutines in VTIP to parse a VHDL structural model of the target architecture built using PML components in order to extract connection information needed by the APT partitioning tool. This tool is also documented in her thesis.
5. APT Spreadsheet to VHDL Performance Model Interface. Dirk Ziegenbein wrote this interface using the standard Unix parsing tool, NAWK. This interface reads an ASCII file produced by the APT tool that describes the software to hardware mapping being studied, reads the file produced by CONET,

and reads the VHDL structural model of the target architecture built using PML components. The interface automatically generates an executable VHDL performance model including the required test bench. The test bench is executed to simulate the performance of the target architecture with the target algorithm mapped onto the architecture by the APT tool. This dynamic performance simulation will find system bottlenecks and under- or over-utilized components.

3.1.2 Test Bench Generator

3.1.2.1 Use of Commercial Tools to Generate Test Benches

The following commercial tools were used in the test bench generation process:

1. Illogix Express VHDL - See Section 2.2.5.2 for a discussion of using state charts with Express VHDL to model IRST test benches and Section 2.2.5.4 for an example of using activity charts with VHDL to model SAR test benches.
2. Cadence Signal Processing Work System (SPW) - See Section 2.2.5.4 for a discussion of modeling SAR with SPW.
3. Synopsys Graphical Environment(SGE) - See Section 2.2.5.5 for a discussion of how SGE was used to construct structural models of test benches. SGE was also used to capture the Specification Repository. See Section 2.2.6.1.
4. RDD 100 was used to capture requirements for IRST. Its use is discussed in Section 2.2.4.

3.1.2.2 Requirements Captures and Use in Test Benches

Synopsys Graphical Environment (SGE) was used to capture the Specification Repository. See Section 2.2.6.1. RDD 100 was used to capture requirements for IRST. Its use is discussed in Section 2.2.4. The system requirements are used to generate values of test bench generics. How this is done is discussed throughout Section 2.2.

3.1.2.3 Use of Configuration Declarations to Reuse Test Bench Components.

VHDL configurations were used to bind the test bench structural architecture to library components, thus promoting code reuse between model years. Details of this are given in Sections 2.2.5.5 and 2.7 and 2.8.

3.2 Library Construction

3.2.1 Conversion of SPW Test Benches to VHDL

(see Section 2.2.5.4)

3.2.2 Use of IRAMP Data Bases

(see Section 2.2.5.7)

3.2.3 IRTOOL Models

(see Section 2.2.5.7)

3.2.4 xpatch Models

(see Section 2.2.5.7)

3.2.5 Use of Honeywell/Omniview Performance Model Library

We used components from the Honeywell/Omniview Performance Model Library, version PML_02a, to construct VHDL performance models. We modified some models and added new ones to satisfy our needs. This library is delivered in subdirectory apt/data/PML_02a. There are three main subdirectories: packages, processors, and leaf_cells.

3.2.5.1 Packages Subdirectory (apt/data/PML_02a/packages)

Subdirectory packages contain package declarations and package bodies for a variety of uses. The original PML packages include packages of constants, packages of routines to collect statistics, etc.

3.2.5.1.1 Additions to the Packages Subdirectory

We added a package of constants called constants.vhdl to the packages subdirectory that includes all of the constants used in our models.

We also added a package of subroutines that are used by the tool that automatically creates the final executable VHDL performance model. These subroutines, described in the following table, are delivered as file subroutines-p.vhdl (package declarations) and file subroutines-b.vhdl (package body).

Table 3. Subroutines and Their Descriptions

Subroutine Name	Description
readp	Sends a read token from a processor to a memory device
writep	Sends a write token from a processor to a memory device
controlp	Sends a control token from one processor to another processor
distributep	Distributes tokens from a single source to multiple destinations on a round robin basis
splitp	Distributes tokens from a single source to multiple destinations on a first available basis
split_initp	Initializes a split operation by starting tasks in each of the destination processors
broadcastp	Broadcasts received tokens to all processors in a set of destination processors
donep	Procedure used to end all tasks
hostcheckp	Procedure used at the beginning of each task to detect errors in task mapping and to verify that the target processor is a valid processor

3.2.5.2 Processor Subdirectory

The processors subdirectory (apt/data/PML_02a/processors) includes all of the original PML processor components needed to develop performance models. These components are delivered under our government use license. Distribution outside the government is not permitted.

3.2.5.2.1 Additions to the Processor Subdirectory

The VTIP analyzer would not recognize certain constructs used in the Honeywell processor models. Therefore, in order to parse the models using the VTIP tool, we were forced to modify certain components in the processor library. These modified components were used only for VTIP parsing, and never for actual VHDL simulation. The modifications were relatively minor and are detailed in Priya Balasubramanian's Masters thesis. To differentiate between the modified models and the original models, we named the modified model files with the same name as the original model file but used an extension of *.vtip instead of the original extension of *.vhdl. The following modified files are delivered in subdirectory processors.

procapplication-a.vtip

processor-c.vtip

3.2.5.3 Leaf_cells Subdirectory

The leaf_cells subdirectory (apt/data/PML_02a/leaf_cells) contains all of the original PML components except the processor components. These components include input devices, output devices, bus interface units, etc.

3.2.5.3.1 Additions to the leaf_cells subdirectory

We needed to modify certain leaf cells for our purposes. We added an h to the filename of each model that we modified to distinguish our modified models from the original models. For example, the original filename comm_int-a.vhdl became comm_inth-a.vhdl. The following models were modified:

comm_inth-a.vhdl
comm_inth-e.vhdl
global_inth-a.vhdl
global_inth-e.vhdl
indeviceh-a.vhdl
indeviceh-e.vhdl
memoryh-a.vhdl
memoryh-e.vhdl

We also added additional components that were not in the original PML_02a library. The following table describes the added components.

Table 4. Added Components and Their Description

Component Name	**Description**
biu_four-a.vhdl biu_four-e.vhdl	Cluster of 4 bus interface units connected to a common bus
biu_star-a.vhdl biu_star-e.vhdl	Cluster of 3 bus interface units connected to a common bus
crossblock-a.vhdl crossblock-e.vhdl	Crossbar component that connects an input port to each output port
crossbar-a.vhdl crossbar-e.vhdl	Six port crossbar consisting of six crossblock components

3.2.5.4 Compilation (Analysis) of Library Components for VHDL Simulation

Prior to initial use, all of the components in the enhanced PML library must be analyzed by the Vantage analyzer. Also, occasionally, these components must be re-analyzed if they become corrupted. The subdirectories must be analyzed in the following order: Packages, Processors, Leaf_cells. Each directory contains an executable script file named vcomp* that can be executed to analyze a single file. The command form is: vcomp filename.vhdl. Also, each directory has an executable script file named van-comp* that can be executed to analyze all of the components in the directory in the proper order. Van-comp calls vcomp repeatedly.

3.2.5.5 Compilation (Analysis) of the Library Components for VTIP Use

Before calling any of the C language subroutines provided by VTIP, all component files must be analyzed by the VTIP analyzer. Again, the files in the three subdirectories must be executed in the following order: Packages, Processors, Leaf_cells. Each directory contains an executable UNIX script file named vtipc* that can be executed to analyze all of the files in the subdirectory in the proper order. The VTIP analyzer creates a new file in the directory apt/data/dls for each file analyzed. The name of the new file is the same as the name of the original VHDL file but with extension *.vhdlview instead of the original extension of *.vhdl. The files in the dls library are used by VTIP subroutine calls to find constructs in the VHDL program.

3.2.6 Use of SPW Libraries to Generate Performance Models

Our initial SPW models were constructed in the normal manner from standard libraries. These models were extensively parameterized to allow rapid functional verification on small test sets. As we commenced construction of the TIL programs for data extraction, we developed our initial candidates for the APT library. These were composite hierarchies of standard library blocks. An example is the FIR filter of a vector input (we named it fir_vector) which required conversion of the input vector to a string, filtering the string, conversion of the output string to a vector, and discarding the "startup" components. We also found the need for copies of a block that represent different functions, such as a vector source to represent sensor input and one to represent a file access. This was first accomplished by constructing a higher level model consisting only of the required block (i.e., an encapsulated block) with TIL attached at the higher level.

As cited in paragraph 3.1.1.3, we found that we were unable to extract control data, essential for performance modeling, with the SPW Tool Interface Language and had to copy the model_name.mseq file which is generated when the functional model is simulated or when it is used to generate a C program. When an extraction is made using TIL, the netlister flattens the hierarchy to the level where it first

encounters a TIL file for that program and considers that to be a leaf block. The model_name.mseq file contains data for the standard library blocks at the bottom of the model. This required modifying our composite blocks to contain two components, a standard block with TIL that extracts nothing and the remainder of the function with TIL that will produce the spreadsheet row. This way, the rate (number if iterations of the function for one iteration of the algorithm) can be captured from the rate of the standard library block as reflected in the model_name.mseq file. With this expansion, we generated running Network II.5 models of the simple model and the single polarity pair SAR.

When we generated a spreadsheet for the full four polarity pair (three pairs processed) SAR, we found it extremely difficult for the user to determine which of three identical functions was associated with which pair. (In the SPW model, the three pairs use the same identical model.) This problem was corrected by adding another parameter to the model named thread. This parameter was set to HH, HV, and VV in the three threads and extracted and placed next to the function in the spreadsheet. Introducing this parameter to all blocks, including those in standard libraries, required us to make a complete copy of all blocks used rather than using some in the standard libraries. While this introduced some redundancy, it removed the need for encapsulated blocks since we could make multiple copies of the same block with different names and TIL attached.

3.2.7 Development of VHDL DSP Library of Primitives and Applications

This library was developed to provide the test bench generator tool with MUTs to test. It consists of several packages and the final MIT SAR Benchmark 1 model. These models were developed by Ram Gummadi. His Master's thesis, Methodology for Structured VHDL Model Development, explains the models and the process used to construct them.

The package DSP_PRIMS contains the following primitive DSP functions.

FFT_SIG	Fast Fourier Transform with signals
FILT_SIG	Filter with signals
CONV_SIG	Convolution with signals
FFT_VAR	Fast Fourier Transform with variables
FILT_VAR	Filter with variables
CONV_VAR	One dimensional convolution with variables
CON_VAR	Two-dimensional convolution with variables

Package IEEE_math contains trigonometric functions with real data types.

Package types_pkg contains declarations of all of the data types.

Package PRIMS contains the major functions in the SAR algorithm, summarized in the following table.

Function Name	Description
VBIQ	Video to Baseband IQ Conversion
RG_COMPR	Range Compression
CORN_TURN	Corner turn operation
AZI_COMPR	Azimuth Compression

3.3 Applications of Test Benches

3.3.1 SAR Test Benches

(see Section 2.2.5.4)

3.3.2 IRST Test Benches

(see Section 2.2.5.1)

3.3.3 SAR Performance Modeling

The RASSP benchmark algorithm and derivatives of it was the primary algorithm employed in developing the APT tool.

3.3.3.1 SPW SAR Algorithm Model

The SPW SAR models, both single polarity pair and the full model, were completed early in the project. Figure 55 is the top-level graph in SPW. Each of the SAR PROCESS and OUTPUT blocks are the same single polarity pair model. The only thing unique to the full model is the LOAD AND DISTRIBUTE block. The three models, apt_example, apt_1_pol, and apt_3_pol, are included with the library, apt_lib, in the apt.dmp file delivered with this report.

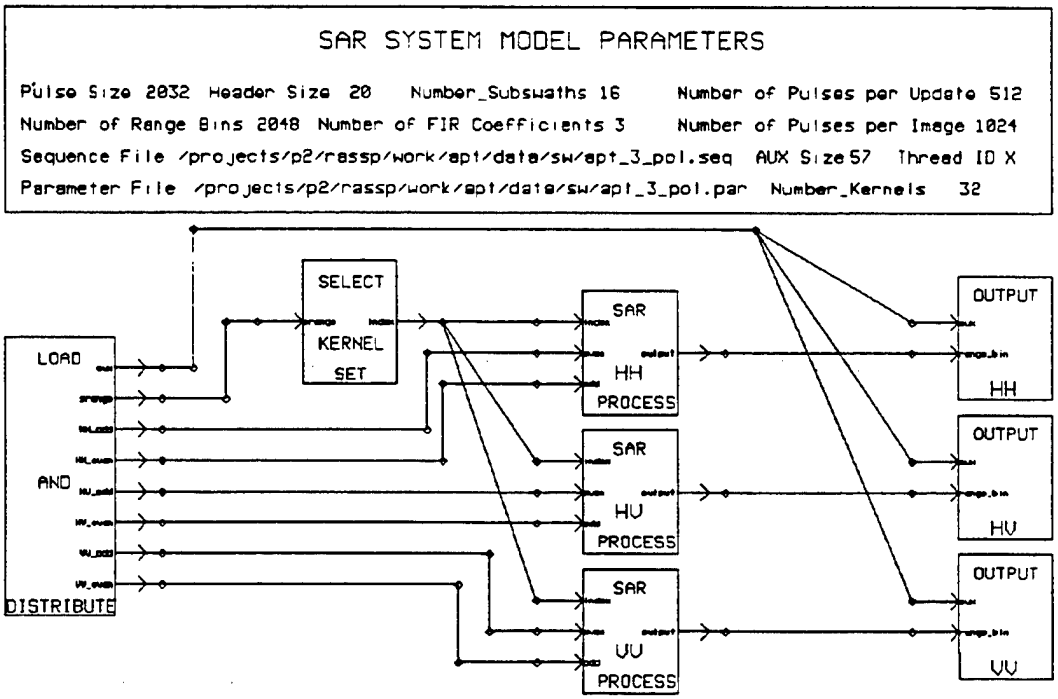


Figure 55. Top-Level Graph in SPW

3.3.3.2 Spreadsheet Hardware Tradeoffs

During the period prior to the first RASSP conference we experimented with numerous spreadsheet layouts. As stated in paragraph 3.1.1.2, initial efforts used a textbook SAR and a notional IRST algorithm. Later in the period we focused on the RASSP benchmark SAR. We discovered that much insight can be gained in early design stages through static analysis comparing processor clock speeds and instruction architectures and analyzing memory requirements. This led to development of sizing as well as mapping spreadsheets, and to the eventual incorporation of the ability to change processor types in a spreadsheet from a menu. These analyses quickly showed that the large memory requirement for the

overlapped corner turn was a critical issue. Results were passed to the prime contractors and to the benchmark contractor. When the Mercury Raceway architecture was selected, the memory issues were amplified. The sizing models allowed rapid comparison of the desired SHARC and fallback i860 processors. The value of the sizing analysis followed by simulation of selected architectures was proven in the JAST analyses reported in the second interim report and summarized in paragraph 3.3.4. Spreadsheets for the

simple model and the one polarity pair generated with Network II.5 and VHDL generated .ssh files along with output for performance model generation are included in the models directory of the delivered software in the 2020 subdirectory in subdirectories race_15 and vpirace16, respectively.

3.3.3.3 VHDL Performance Models

We are delivering one VHDL architecture model located in directory apt/models/vpirace16. This is a 16-processor raceway architecture. There are three files: 1) vpirace16.vhdl is a VHDL structural model of the architecture using components from the enhanced PML library, .2) vpirace16-c.vhdl is a configuration file with all of the hooks needed to map software tasks to the processors. However, there are no software tasks specified. Each processor is assigned an idle task that will be replaced by application tasks by the automated APT tool, and 3) vpirace16.vtip is the modified version of the architecture file suitable for VTIP analysis.

Subdirectory range16 (apt/models/VHDL/vpirace16/range16) contains the output of the APT tools when the range processing and corner turn segments of the MIT SAR Benchmark algorithm are mapped onto the architecture.

Subdirectory apt/data/PML_02a/test contains the executable file simvpirace16* that will execute the VHDL performance model created by the APT tool.

3.3.3.4 BONeS Performance Model

As stated in paragraph 3.1.1.2, before we abandoned BONeS as the performance modeling tool we completed BONeS models of our three versions mapped onto arrays of TMS320C40 processors. Figures 56 and 57 show the top level and the range compression graph for the full SAR model. The BONeS models are included in the models directory of the delivered software in the bones subdirectory.

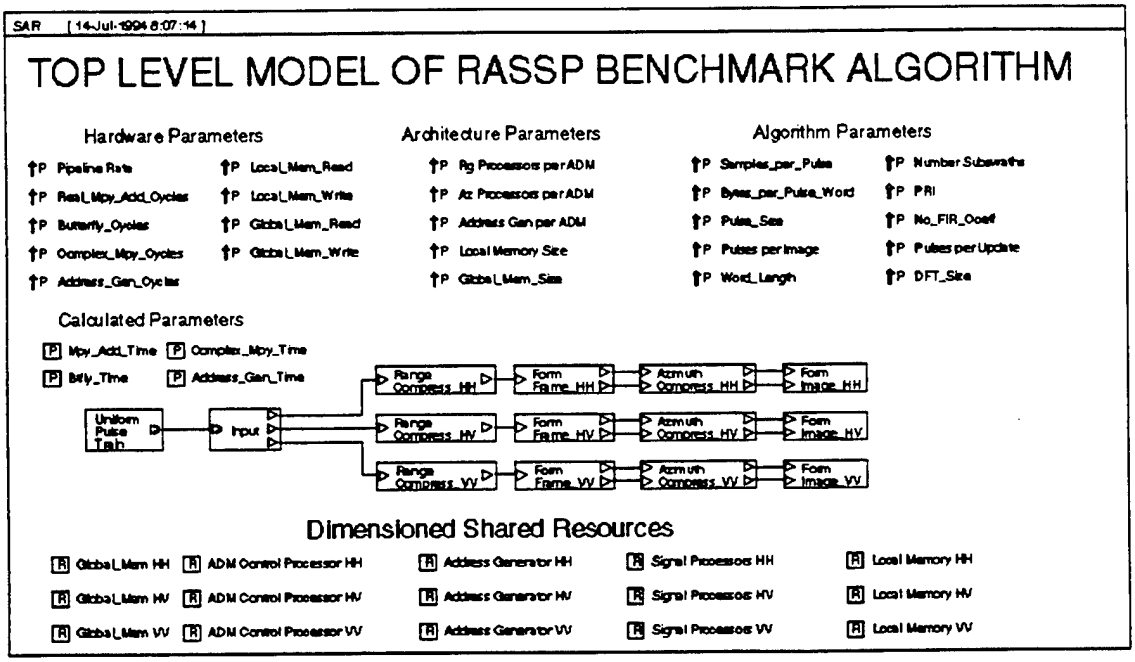


Figure 56. Top-Level Model of RASSP Benchmark Algorithm

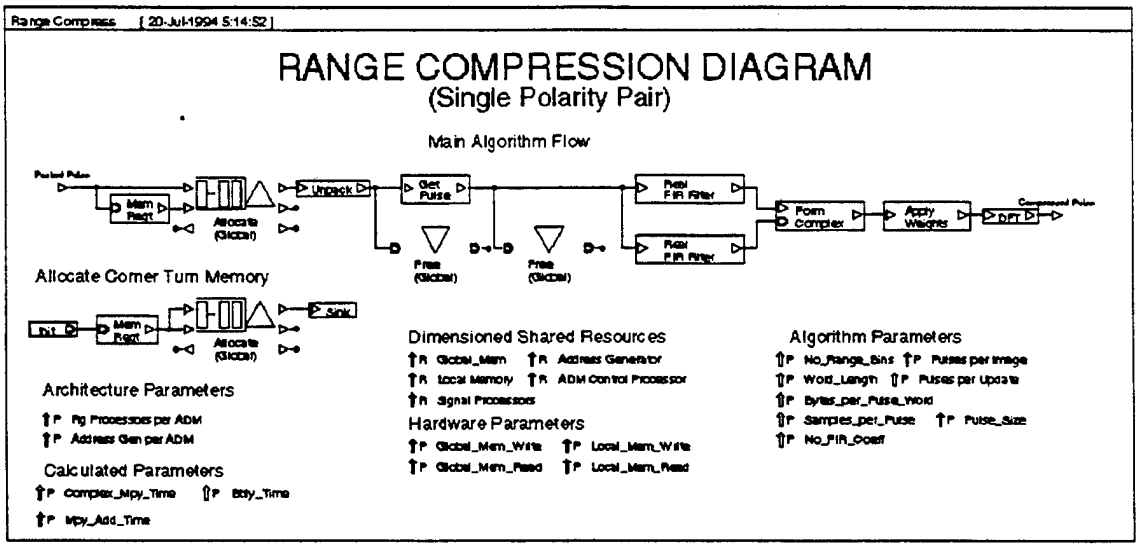


Figure 57. Range Compression Diagram

3.3.3.5 Network II.5 Performance Models

Target performance models were constructed for all three target algorithms mapped onto the Mercury Raceway architecture. The model for the full SAR algorithm is included in the models directory of the delivered software in the N25/target subdirectory. Schematics for three architecture variations are included in the models directory of the delivered software in the N25/schem subdirectory. Generated models of the 1_pol and simple models mapped to the race_15 architecture are included in the models directory of the delivered software in the N25/generated subdirectory. All three performance models are accompanied by a .lis file containing standard output from simulation. Figure 58 is a printout of the network model.

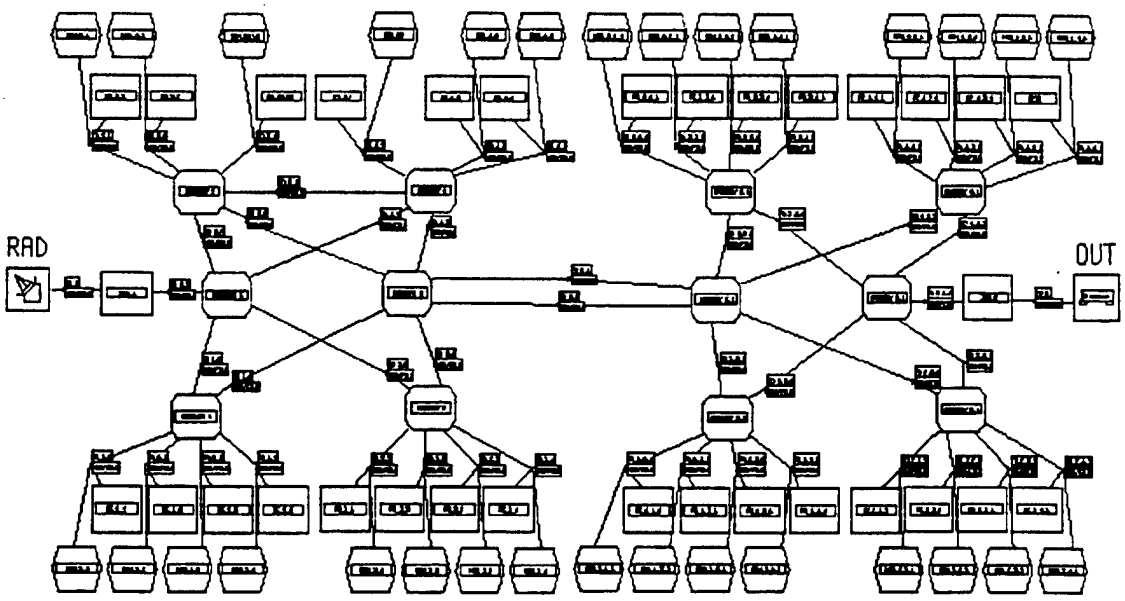


Figure 58. Network Model

3.3.4 JAST Performance Modeling

As stated in paragraph 1.3, the benchmarking effort at Northrop-Grumman commenced with analyzing the requirements and performance of the RASSP SAR algorithm on their candidate architectures, discarding non-contenders and then analyzing additional algorithms on the remaining architecture variations. We also did some initial modeling of a Real Beam Ground Map (RGBM) algorithm [RGBM].

3.3.4.1 Tradeoff Space Explored

The basic architecture consisted of signal processing groups consisting of four DSPs, memory, and an interface. We varied processor characteristics over four different DSP instruction sets and clock rates from 25 to 200 MHz. We analyzed variations with local DSP RAM, Global RAM for the array, and both. We examined different bus structures and characteristics for access to local and global memories.

3.3.4.2 Use of Spreadsheet to Narrow the Design Space

A series of sizing spreadsheet studies were conducted on four radar algorithms with various instruction sets and clock rates. These reduced the set of processor characteristics to be considered to a few and provided estimates on processing and memory requirements.

3.3.4.3 Use of Performance Model to Identify Bottlenecks

Extensive trade studies were conducted using dynamic performance models of the RASSP SAR algorithm on numerous variants of the JAST architecture. These studies addressed questions provided by weapon system contractor system engineers.

The RASSP SAR algorithm can be considered as three groups of functions that are processed in parallel, one group for each of the three polarity pairs. Each polarity pair is composed of a range processing and an azimuth processing segment separated by a matrix transpose (corner turn) operation. Mappings were investigated with each of three signal processing groups processing a single polarity pair and with all of the processing conducted in two groups. Within these approaches, we simulated mapping the range processing on a single DSP and the azimuth processing on both a single DSP and on two DSPs. The latter mapping roughly equalized the processor utilization on the range and azimuth processors.

Memory requirements provided to be a key issue in the trade studies leading to a shift away from the initial processor/architecture. Some of the initial memory organization variations that we considered were DSPs with and without local memories.

Studies were conducted of required bus widths and rates in conjunction with memory/cache combinations. Even though the utilization of the buses was well below 100%, contention for the buses caused the loss of some incoming pulses in the initial simulation of the no local memory alternative. This necessitated a

complex hierarchy of priorities and interrupt restarts to ensure all pulses and range bins were processed. This loss of data indicated that the design was close to saturation. This was an indication of a high risk that this design, when fully implemented, would not be able to meet the processing requirements of the algorithm.

4. Future Directions

The APT toolset is a "proof of concept" system. Employment by JSF avionics system engineers to analyze concepts proved its value. However much improvement is required to evolve a fully commercialized toolset.

Among these are the following extensions of the toolset:

Extending the seamless design environment. The SPW tool set can generate codes for a single processor. Combining the APT capabilities with the SPW code generators would allow the partitioning to be done once in the resource utilization environment and then have the code generated for the target multiprocessor system automatically.

Integration of the resource utilization analysis tools with man in the loop (MITL) simulations. The JSF project started efforts to link the performance models generated by the APT to high load segments of MITL simulation traces.

Integration of the signal processor resource utilization analysis tools with the system architecture analysis tools used in high level trade studies. The APT system can be used to generate detailed timing estimates for the signal processing modules that can be factored into the complete avionics architecture simulations.

Developing a complete suite of benchmark algorithms and associated library expansion. This suite should encompass memory demanding algorithms such as the SAR algorithms used in the RASSP and JSF efforts with processing demanding algorithms such as multiple PRF search-while-track algorithms.

Validating the models against actual systems. The development of the APT system has focused on the verification issue, developing tools that automatically provide traceability of performance model events and parameters back to algorithm primitives and parameters and hardware structures and parameters. In order to increase the comfort level in the simulations and analyses, a model of existing software and hardware must be generated and the timing of the performance model should be compared with timing of the actual system. The timing measures of the SPW primitives used in the spreadsheet have been compared with timings for the same primitives provided in hardware manufacture application notes.

Developing a graphical user interface to activate the various tools and select input sources and output locations.

In addition several revisions would be in order. These are associated with the lessons learned described in paragraph 5.

Rewrite the tools for extraction of data from the VHDL structural model in awk. This would eliminate the need for VTIP and the attendant need to develop a separate model for extraction. The new extraction routines would extract only the required data.

Modify the target models and the VHDL to spreadsheet and spreadsheet to VHDL interfaces to employ the latest (commercialized) version of PML.

The current version of the bld_route program uses Moore's Algorithm to compute one shortest path (corresponding to the route) between each pair of components. Because the components are ordered, the network traffic might not be distributed well enough. The Algorithm should be

modified to compute all shortest paths (routes) between each pair of components and select a route randomly or to select routes during simulation based on real traffic.

A test plan can be combined with goal trees to develop a high approach to system testing. A goal tree will represent a test plan. The goal tree is given a grand goal such as "determine the system noise sensitivity." The grand goal is decomposed into subgoals until primitive goals are reached. Primitive goals define a test group that would be applied to reach the grand goal.

5. Lessons Learned

Several lessons were learned in the course of this project.

Building target models using an under development library requires much time devoted to library debugging. This led to completion of extraction programs before working target models were completed. The result is that much unnecessary data was extracted from the structural model and one item (INST = PML instantiation name) that should have been extracted was not. Correction of this would eliminate artificial constraints on component names in the structural model.

VTIP was harder to learn than expected. It also could not analyze PML files as delivered. This required modified PML files and a redundant structural model. In retrospect, direct extraction of data from the VHDL files with awk would be more efficient and would provide an easier to use toolset.

In our RASSP sponsored work, we have shown that effective test bench generation requires:

1. High-level graphics design tools to develop initial test bench code quickly.
2. A test bench component library to construct test benches structurally.
3. Accurate environmental modeling using application domain specific software.
4. Automatic linkage to the system specification.
5. A test plan interface to configure the test bench structural model.

And while commercial software can be used for generation of test bench pieces, system software having graphic user interfaces is needed to integrate the pieces into a working system.

6. Published Papers

1. James R. Armstrong, Geoffrey Frank, Srinivasan Hrishikesh, Prabhakar Gowrisankaran, Zhen Xu, "Test Bench Development for RASSP DSP Models," Proceedings of the First Annual RASSP Conference, Arlington, VA, August 15-18, 1994, pp. 91-96.

2. R. Armstrong, G. Frank, et al., "High Level Generation of VHDL Test Benches," Proceedings of the Spring VHDL International Users Forum, April 1995, pp. 6.23-6.34.

3. A. Frank, J. R. Armstrong, and F. G Gray, "Support for Model Year Upgrades in VHDL Test Benches," Proceedings of the 2nd Annual RASSP Conference, July 1995.

4. Gray and J. R. Armstrong, "Reutilizaton of VHDL Test bench and Library Components," Proceedings of the 1994 American Institute of Aeronautics and Astronautics., San Antonio, Texas, March 28-30, 1995, pp. 691-700.

5. R. Armstrong, G. A. Frank, and F. G. Gray, "Efficient Approaches to Testing VHDL DSP Models," Short Paper Presented to ICASSP 95. Full journal article to appear in the Journal on VSLI Signal Processing.

6. G. A. Frank, B. E. Clark, and W. G. Ransdell, "Adapting algorithms to architectures through transformations," *Proceedings of 1st Annual RASSP Conference*, August 15-18, 1994, pp. 171-178.

7. H. P. Commissariat, F. G. Gray, J. R. Armstrong, G. A. Frank, "Developing Reusable Performance Models for Rapid Evaluation of Computer Architectures Running DSP Algorithms," *Proceedings of the 2nd Annual RASSP Conference*, July 24-27, 1995, pp. 103-108.

8. G. A. Frank and B. E. Clark of Research Triangle Institute, and B. Schaming and W. Kline of Lockheed Martin Advanced Technology Laboratory. "Hardware/Software Codesign from the RASSP Perspective." Journal article to appear in the Journal of VLSI Signal Processing.

7. Masters Thesis

1.) P. Gowrisankaran, "Structural test bench development for DSP models," Masters Thesis , Bradley Department of Electrical Engineering, Blacksburg, VA, March 1995.

2.) S. Hrishikesh, "Behavioral Test Bench Development For DSP Models," Masters Thesis , Bradley Department of Electrical Engineering, Blacksburg, VA, March 1995.

3.) Z. Xu, "Modeling SAR signals and sensors using VHDL," Masters Thesis, Bradley Department of Electrical Engineering, Blacksburg, VA, February 1995.

4.) Sailesh Kottapalli, "A Test Plan Driven Test Bench Generation System" Masters Thesis, Bradley Department of Electrical Engineering, Blacksburg, VA, July 1996.

References

[Are95] Arete' Associates, IRTOOL Reference Manual, February 1995.

[ArmJ93] J.R. Armstrong and F.G. Gray, Structured Logic Design With VHDL, Prentice Hall, Englewood Cliffs, NJ, 1993.

[ArmJ94] James R. Armstrong, Geoffrey Frank, Srinivasan Hrishikesh, Prabhakar Gowrisankaran, Zhen Xu, "Test Bench Development for RASSP DSP Models," Proceedings of the First Annual RASSP Conference, Arlington, VA, August 15-18, 1994, pp. 91-96.

[ArmJ95] J. R. Armstrong, G. Frank, et al., "High Level Generation of VHDL Test Benches," Proceedings of the Spring VHDL International Users Forum, April 1995, pp. 6.23-6.34.

[ArmJ96] J. R. Armstrong, G. A. Frank, and F. G. Gray, "Efficient Approaches to Testing VHDL DSP Models," Short Paper Presented to ICASSP 95. Full journal article to appear in the Journal on VSLI Signal Processing.

[Asc] Ascent Logic Corporation, RDD-100 User's Guide, Release 4.

[BalA94]Balboni, A., Mastretti, M., and Stefanoni, M., "Static Analysis for VHDL model Evaluation," EURO VHDL, 1994.

[Cad94] Cadence, Inc., SPW-The DSP Framework, User's Guide, March 1994.

[CamS93] S. Campana, "Passive Electro-optical Systems," The Infrared & Electro-Optical Systems Handbook, ERIM, Ann Arbor, MI, 1993.

[ChoC94] C. H. Cho and J.R. Armstrong, "The B Algorithm: A Behavioral Test Generation Algorithm," Proceedings of the International Test Conference, Fall 1994, pp. 968-979.

[Com92] Comdisco Systems, Inc., Signal Processing WorkSystem/The DSP Framework, 1992.

[Dem93] Demaco, Inc., User Manual for x-patch, September 15, 1993.

[EurA93] Proceedings of EUROASIC 93.

[FraG91] G. A. Frank, "The Evolution of External Models For Simulation," Proceedings of the Systems Design Synthesis Technology Workshop, Naval Surface Warfare Center, Silver Spring, MD, Sept. 10-13, 1991.

[FraG95] G. A. Frank, J. R. Armstrong, and F. G Gray, "Support for Model Year Upgrades in VHDL Test Benches," Proceedings of the 2nd Annual RASSP Conference, July 1995.

[GowP95] P. Gowrisankaran, "Structural test bench development for DSP models," Masters Thesis , Bradley Department of Electrical Engineering, Blacksburg, VA, March 1995.

[GraG94] F. G. Gray and J. R. Armstrong, "Reutilizaton of VHDL Test bench and Library Components," Proceedings of the 1994 American Institute of Aeronautics and Astronautics., San Antonio, TX, March 28-30, 1995, pp. 691-700.

[GajD94] D. Gajski, F. Vahid, S. Narayan and J. Gong, Specification and Design of Embedded Systems, Prentice Hall, NJ, 1994.

[HarD87] D. Harel, "STATECHARTS: A Visual Formalism for Complex Systems," Science of Computer Programming 8, 231-274, North-Holland, 1987.

[HriS95] S. Hrishikesh, "Behavioral Test Bench Development For DSP Models," Masters Thesis , Bradley Department of Electrical Engineering, Blacksburg, VA, March 1995.

[IEEE88] IEEE Standard VHDL Language Reference Manual, IEEE, New York, 1988.

[Ilog92] i-Logix, ExpressV-HDL Reference Manual, i-Logix, Inc., Burlington, MA, Vol. I, Version 3.0, December 1993.

[KapS94] S. Kapoor, J. R. Armstrong, and S. R. Rao, "An Automatic Test Bench Generation System," Proceedings of the VHDL International Users Forum, Spring 1994, pp. 8-17.

[KotS96] Sailesh Kottapalli, "A Test Plan Driven Test Bench Generation System," Masters Thesis, Bradley Department of Electrical Engineering, Blacksburg, VA, July 1996.

[LiW96]W. Li and J. R. Armstrong, "Test Generation from VHDL Behavioral Models," Proceedings of the VHDL International Users Forum, Fall 1996.

[Mat92] The Math Works Inc., MATLAB Reference Guide, Oct., 1992.

[ONR94] Office of Naval Research, "Infrared Analysis Measurements and Modeling Program-Data Catalog," January 1994.

[PauB92] Paulsen, B. and Levia, O., "Techniques for Writing High Performance and High Quality VHDL Models," EURO VHDL, 1992.

[RicM94] M. A. Richards, "The RASSP Program: Overview and Accomplishments," Proceedings of the 1st Annual RASSP Conference, Arlington VA, August 15-18, 1994, pp. 1-8.

[RBGM] AFWAL-TR-86-1017 "Generic Signal Processor Architecture (GSPA)" Vol. I "Requirements" AT&T Bell Laboratories February 1986.

[ShaG94] G. A. Shaw, "Synthetic Aperture Radar Image Processor - RASSP Benchmark," M.I.T. Lincoln Laboratory, January 1994.

[Syn92] Synopsys Graphical Environment User Guide, Version 3.0, December 1992.

[Syn95] Synopsys VHDL System Simulator - Command Reference Manual, Version 3.3b, Sept. 1995.

[Van92] Vantage Analysis Systems, Inc., Vantage Spreadsheet User's Guide Volumes I and II. 1992.

[Wav92] IEEE Standard 1029.1-1992- Wave Form and Vector Exchange Specification.

[WicJ96] Wicks, J. A., Jr. and Armstrong, James, R., "VHDL Model Efficiency," Asian Pacific Conference on Hardware Description Languages, Bangalore, India, January 1996.

[XuZ95] Z. Xu, "Modeling SAR Signals and Sensors Using VHDL," Masters Thesis, Bradley Department of Electrical Engineering, Blacksburg, VA, February 1995.

[ZueB94] B. Zuerndorfer and G.A. Shaw, "SAR Processing for RASSP Application," Proceedings of the 1st Annual RASSP Conference, Arlington VA, August 15-18, 1994, pp. 253-268.