

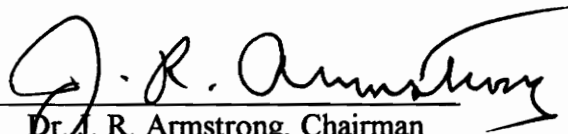
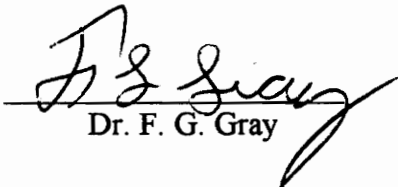
Structural Testbench Development for DSP Models

by

Prabhakar Gowrisankaran

Thesis submitted to the faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
Master of Science
in
Electrical Engineering

APPROVED:


Dr. J. R. Armstrong, Chairman
Dr. F. G. Gray
Dr. Peter M. Athanas

March 1995

Blacksburg, Virginia

C.2

LD
B655
V855
1995
G697
C.2

Structural Testbench Development for DSP Models

by

Prabhakar Gowrisankaran

Dr. James R. Armstrong, Chairman

Electrical Engineering

(ABSTRACT)

Generation of test benches for large DSP behavioral models is a complicated, labor intensive task. Also, tests generated manually satisfy no formal definition of completeness. To address these needs, high level approaches to test bench development are employed which relieve the modeler of the details of test bench development.

Two approaches being explored are: 1) behavioral - the CASE tools develop complete high level models of the test bench, and 2) structural - a library of primitive components is developed so that a conventional schematic capture tool, e.g., Synopsys Graphical Environment, can be used to construct the test bench. An intelligent interface prompts the user for high level test bench information, and inserts this information into the test bench code. The intelligent interface also allows the user to specify and control file I/O as a data source.

The objective of this thesis is the development of a set of library models from which a structural test bench can be created. This thesis also describes creation of the structural test bench using the library of primitive components and Synopsys Graphical Environment, a conventional schematic capture tool. This approach has been implemented

for two applications: 1) A 2D Image processing algorithm - InfraRed Search and Track (IRST) and 2) Synthetic Aperture Radar (SAR). An intelligent interface developed in C combines the structural model with user information which provides for generics and inputs. This thesis also shows how a requirements capture tool can be used to generate generic values.

To Amma and Appa

Acknowledgments

I would like to express my sincere thanks to my advisor Dr. James R. Armstrong for providing constant support and guidance throughout my thesis research. It has been an honor and a pleasure to work in a challenging environment under him. I would also like to express my gratitude to Dr. F. Gail Gray for his support as a part of the RASSP team and for serving as a member on my committee.

Also, I would like to thank Dr. Peter.M.Athanas for being on my graduate committee. I would also like to thank Dr. Geoffrey A. Frank and Dr. Bud Clark of Research Triangle Institute, Raleigh for their support through the course of this research.

I convey my deepest appreciation to all my friends. I appreciate the motivating company of my room mates, Hrishi and Ganesh. I would like to dedicate this work to my parents who have always encouraged me at all the times. Their unreserved love and support are the main reasons for all my achievements. I would also like to thank my brother Siva and my sister in-law Soumya for their encouragement.

Table of Contents

Chapter 1. Introduction 1

 1.1 Motivation 1

 1.2 The RASSP Test Bench Generation Task 3

 1.3 Research Approach 4

 1.4 Thesis Organization 6

Chapter 2. Background 8

 2.1 Overview on Test Benches 8

 2.2 Test Vector Generation Techniques 11

 2.3 Test Bench Applications 12

 2.4 Test Bench Generation System 13

Chapter 3. RASSP Specific Applications 15

 3.1 Introduction 15

3.2 IRST Test Bench	16
3.2.1 IRST Stimulus Generator	16
3.3 SAR Test Bench	19
Chapter 4. Requirements Capture	22
4.1 Introduction	22
4.2 Requirements Interface for IRST	24
4.2.1 Primary and Secondary Requirements for IRST	24
4.2.2 Math model for the Requirements Interface	26
4.2.3 Realization of the Math Model using RDD-100	27
Chapter 5. Library Primitives	32
5.1 Overview	32
5.2 Features of Synopsys Graphical Environment (SGE)	33
5.3 Development of Library Primitives	34
5.3.1 Primitives for IRST Test Bench	34
5.3.2 Primitives for SAR Test Bench	38
5.4 Parameterization of the Primitives	44
Chapter 6. Structural Model Construction	52
6.1 Methodology	52
6.2 Structural Test Bench Generation	53
6.2.1 Infrared Search and Track Test Bench	53
6.2.2 Synthetic Aperture Radar Test Bench	55
6.3 Procedure based Test Bench Model for IRST	59

Chapter 7. Intelligent User Interface 60

 7.1 Purpose 60

 7.2 Menu Structure 61

Chapter 8. Results 67

 8.1 Simulation Results 67

 8.2 Output Displays 69

 8.2.1 IRST Test Bench 69

 8.2.2 SAR Test Bench 73

Chapter 9. Conclusions and Future work 74

Bibliography 76

Appendix A: Behavior Diagram for Time Function Requirements 78

Appendix B: VHDL Code listing for IRST and SAR 80

Appendix C: Code Listing for IUI 115

Appendix D: Simulation Control Files 139

Appendix E: Code Listing for C program to measure CPU time 145

Vita 148

List of Illustrations

Figure 2.1. A Basic Test Bench	9
Figure 2.2. Test Bench Generation System	14
Figure 3.1. IRST Stimulus Generator	17
Figure 3.2. SAR Testbench	20
Figure 4.1. A High Level Test Bench Generation System	23
Figure 4.2. Requirements Interface for IRST	25
Figure 4.3. Derivation of Formulae for Clutter and Target Motion	27
Figure 4.4. Time Function Requirements	29
Figure 5.1. SGE Symbol for Osc	35
Figure 5.2. SGE Symbol for Environ	36
Figure 5.3. SGE Symbol for Target	37
Figure 5.4. SGE Symbol for Sensor	38
Figure 5.5. SGE Symbol for Chirp	39
Figure 5.6. SGE Symbol for Comptfp	39
Figure 5.7. SGE Symbol for Compmul	40

Figure 5.8. SGE Symbol for Genchirp	40
Figure 5.9. SGE Schematic for Genchirp	41
Figure 5.10. SGE Symbol for Delay	41
Figure 5.11. SGE Symbol for Neg	42
Figure 5.12. SGE Symbol for Decimate	42
Figure 5.13. SGE Symbol for Change	43
Figure 5.14. SGE Symbol for Pass	43
Figure 5.15. VHDL Code for TARGET Module	51
Figure 6.1. SGE Symbol for IRST Testbench	53
Figure 6.2. SGE Schematic for IRST Testbench	54
Figure 6.3. SGE Symbol for SAR Testbench	55
Figure 6.4. SGE Schematic for SAR Testbench	56
Figure 7.1. Menu Structure in IUI	62
Figure 7.2. A Template for SCL File	66
Figure 8.1. Display of Output Frame using Synopsys	70
Figure 8.2. Display of Output Frame using MATLAB	71
Figure 8.3. Display of Sequence of Frames using MATLAB	72
Figure 8.4. MATLAB Plot of the SAR Output Signal	73

List of Tables

Table 8.1. Table Depicting CPU Times Recorded for Simulating the Two IRST
Testbench Models 68

Chapter 1. Introduction

1.1 Motivation

The field of VLSI design has undergone rapid technological changes and the complexity of circuitry that can be built on a single chip has been growing at an enormous rate, i.e. , number of transistors that can be built on a chip has reached the millions. This increase in complexity has resulted in the design of a chip at the transistor level being virtually impossible [1]. The duration of the product development cycle, from forming the concept, to fabricating the chip on silicon is also important and has to be kept short to maintain competitiveness. Architectural level iterations which are time-consuming have become unacceptable [2]. So, designers have had to follow various other approaches to design chips, and their needs have been matched by advancements in design automation, resulting in the availability of a number of software tools which make the design cycle quicker and easier. These software tools make it possible to simulate and synthesize designs at abstraction levels that are close to human reasoning [3]. Hardware description

Chapter 1. Introduction

languages like the VHSIC (Very High Speed Integrated Circuit) Hardware Description Language (VHDL) have played an important role in this transformation.

HDLs allow designers to create models of chips at various levels of the design hierarchy ranging from the system level to the switch level [4]. HDLs can model concurrency in the execution of the logic circuits and can also model the timing relationship between the logic blocks. HDLs provide a convenient tool for the functional or behavioral specification and simulation of digital circuits, reducing the quantity and detail of the information which the designer must manage.

The VHSIC Hardware Description Language (VHDL) is becoming extremely popular in electronic industry as a hardware description language. VHDL is an IEEE standardized language (IEEE 1076 standard) [5]. It has a rich set of constructs for circuit modeling, and circuits can be described in either top-down or bottom-up fashion through varying levels of abstraction. After the VHDL behavioral model of a circuit is developed, test data can be generated for simulation of the model. By observing the simulation output, the functionality of the model can be verified.

When designing highly integrated circuits, the significance of the design verification phase cannot be over-emphasized. The general practice in the design verification process is to simulate the circuit with a very large amount of test data given by a designer or a modeler. The generation of this test data is a very time-consuming and a labor-intensive task [6]. The traditional test pattern generation techniques have been rendered inadequate because of the increased complexity and the lack of information about the gate-level description. Also, the test patterns constructed by a modeler may not

Chapter 1. Introduction

test all the operations of a circuit. In relieving the modeler of such problems, use of high-level test generation techniques has become very popular. The important factors in simulation are the ability to accurately describe the functionality of a circuit and the ability to test the different operations by generating appropriate test stimuli. Thus in order to tackle the weaknesses of the traditional test pattern generators, designers generally try to generate tests using a higher level of abstraction, preferably a high-level functional or behavioral description of the circuit.

A major focus of the RASSP program here at Virginia Tech is the development of VHDL models for signal processing algorithms and hardware. Like all simulation models, they must be thoroughly exercised to prove that they exhibit the correct behavior. However, development of model tests and the corresponding VHDL test bench is a labor intensive task. Moreover, model tests generated manually satisfy no formal definition of completeness. Thus there is a critical need to develop high level approaches to development of test benches that relieve the modeler of the details of this time consuming process.

1.2 RASSP Testbench Generation Task

The Rapid Prototyping of Application Specific Signal Processors (RASSP) program is an ARPA/Tri-Service initiative intended to dramatically improve the process by which complex digital systems, particularly embedded digital signal processors, are designed, manufactured, upgraded, and supported [7].

Chapter 1. Introduction

The basic approach to test bench development that has been developed as a part of the RASSP program has the following features:

- 1) Test bench code elements are developed using high level CASE tools. Specific to the RASSP environment, we use Ilogix Express VHDL to specify state machine behavior and Cadence/Comdisco SPW to specify signal processing behavior.
- 2) During model simulation the test bench is under control of an intelligent interface which provides the test bench with values for test bench generics and specifies the high level characteristics of model inputs or controls the reading of files of data reflecting the nature of those same input characteristics.
- 3) Test benches can be developed either behaviorally or structurally. With behavioral development, the CASE tools develop complete high level models of the test bench. With structural development, a library of primitive test bench components and schematic capture tool is used to construct the test bench.

1.3 Research Approach

The primary objective of this thesis is to develop a methodology for structural test bench generation for Digital Signal Processing (DSP) based systems modeled in VHDL. The approach lays emphasis on automating the structural test bench generation task, so as to relieve the modeler of the details of test bench development. The other contributions made by the thesis include the following:

Chapter 1. Introduction

Development of testbench requirements using RDD-100, as one of the goals of the RASSP testbench generation task is to have the testbench to be an immediate interpretation of the specification.

Development of testbench library models using Synopsys Graphical Environment (SGE), a schematic capture tool, thus populating the library of testbench primitives. The use of a schematic capture tool for development of testbench primitives facilitates the automation of the testbench generation task. It also facilitates the reuse of the testbench primitives.

Interconnection of the testbench library primitives developed using SGE, a commercial schematic capture tool to form structural models of the testbench. The use of SGE facilitates the reuse of the testbench structure.

Development of an Intelligent User Interface (IUI) in C. Given all the VHDL components, the IUI configures the testbench, generates data for the testbench inputs and integrates the testbench model into one system. The IUI also forms the simulation control file which is used to run the Synopsys simulator and forwards the simulation results obtained from the testbench to the Model Under Test (MUT).

Development of procedural equivalents of the library primitives. This facilitates comparison of CPU time taken for simulation of the entity based testbench model and the procedure based testbench model. The CPU times were measured using a C program.

Chapter 1. Introduction

The components of the testbench generation system, the requirements interface, the testbench model generator and the IUI were integrated together to form an overall testbench generation system. The testbench generation system was developed on a SUN SPARCstation2 network.

1.4 Thesis Organization

This thesis is organized into an abstract , 9 chapters, bibliography and appendices.

Chapter 1, "Introduction", provides the motivation for this thesis, an overview of the RASSP testbench generation task and an introduction to the work reported by this thesis.

Chapter 2, "Background", provides an overview on testbenches, discusses test bench generation requirements, test vector generation techniques and features of a structural testbench.

Chapter 3, "RASSP specific applications", overviews and discusses the two DSP applications - 2D Image processing algorithm, Infrared Sensing and Tracking (IRST) algorithm and Synthetic Aperture Radar (SAR) for which structural testbenches have been developed.

Chapter 4, "Requirements Capture", discusses the development of the requirement interface for IRST using RDD-100.

Chapter 1. Introduction

Chapter 5, "Library primitives", discusses the development of the library of testbench primitives and the characteristics of the library primitives.

Chapter 6, "Structural Model Construction", discusses structural testbench generation for IRST and SAR.

Chapter 7, "Intelligent User Interface", overviews the development of the intelligent user interface.

Chapter 8, "Results", provides the simulation results of the testbench models and gives a brief description of the outputs obtained.

Chapter 9, "Conclusions and Future work", draws conclusions regarding the methodology adopted and on the implementation of the testbenches and suggests areas for future work.

Chapter 2. Background

2.1 Overview on Test Benches

Test Bench An executable VHDL model which instantiates a model under test (MUT) and provides the capability to drive the MUT with a set of test vectors and compare its response with the expected response [8].

A test bench is the top level in the model hierarchy. It is the entity that is simulated when testing the model [8]. Figure 2.1 shows a basic testbench. The stimulus generator drives the Module Under Test(MUT) with test vectors. The module response is compared with the expected response by the comparator and appropriate Go/No Go signals are generated. When the Model Under Test is a VHDL model, the test bench is also coded in VHDL. This code applies inputs to the MUT using signal assignment statements or file I/O. In addition, state machine behavior must be coded if the Stimulus Generator interacts with the MUT [8].

Chapter 2 Background

Two types of feedback are possible. First, the Module Under Test can feed its state back to the Stimulus Generator, allowing interaction between these two elements. Second, Comparator signals can be fed back to allow adaptive testing, i.e., changing of test modes when a certain response has been observed.

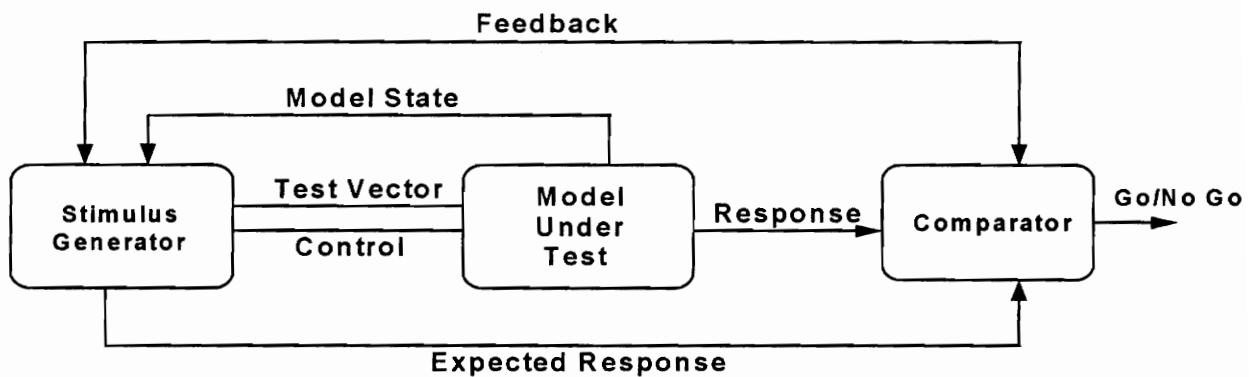


Figure 2.1: A Basic Test Bench

The test bench generator will support at least three different testbench configurations: off-line, on-line, and adaptive. These configurations are described below.

1) Off-line Test Benches

In an off-line test bench, the stimulus generator reads a file of data and at the appropriate time sends the data through the interface to the signal processor. There may be some reformatting done in this process. For example, the input data file could contain

Chapter 2 Background

2-dimensional image stored as 8-bit pixels. The stimulus generator would send these pixels to the MUT in raster scan order [8]. In an off-line test bench configuration the comparator also reads expected responses from a file, and alerts the user if the actual responses don't match the expected responses based on some definition of "match".

If an off-line test bench configuration is used, then the Test Bench Generator (TBG) shall assist the user in selecting data files that are appropriate to the test. These files may be real data from the system being modeled or data files developed with a commercial modeling tool.

2) On-Line, Nonadaptive Test Benches

In this configuration, back-to-back testing is done with a "gold" model of the MUT being run in parallel with the MUT. The data produced by the stimulus generator is copied to both a gold model of the behavior of the MUT, i.e., a model that is assumed to provide the correct responses to stimuli, and the MUT. Their responses can be compared immediately on-line, or stored in output files and compared later off-line.

3) Adaptive Test Benches

Adaptive test benches are the most complex types of test benches. These test benches compute the stimuli during the simulation by taking into account the actions of the MUT, i.e., they adapt their output to match the state of the MUT.

2.2 Test Vector Generation Techniques

Testing of a system is defined as an experiment in which the system is exercised and its response is analyzed to determine whether it is behaving correctly or not [9]. *Test generation* (TG) is defined as the process of determining the test stimuli necessary to test a digital system [9]. Test generation for design verification involves a large effort that, unfortunately is still a manual activity.

The two basic approaches to test vector generation are:

- 1) *White Box approach*: This is a model based approach. The VHDL Model Under Test (MUT) is itself used to produce the tests. The two types of tests generated by this approach are - model perturbation with high level faults [10] and I/O path sensitization [11]. In case of I/O path sensitization the tests are generated so as to activate each of the input-output paths atleast once. Automatic Test Generation (ATG) is an example of model perturbation with low level "*stuck at*" faults.
- 2) *Black Box approach*: This is an environment based approach. This approach uses the environment surrounding the model to produce test vectors. The environment is modeled as a part of the test bench to generate the normal system inputs. This approach is used in the RASSP research.

The test benches developed as a part of this thesis use the environment based approach, rather than a model based approach as the inputs to the Model Under Test are a complicated function of the model environment [12] and could not be directly generated from the model itself.

2.3 Test Bench Applications

A test bench provides the implementation of a test [8]. During the development and fielded use of a system, tests will be applied to the system and to its design. These tests serve several purposes. Some of the purposes are:

1) Behavioral Verification: A test may be applied to determine if the behavior of a system meets the system requirements, where the behavior of a system includes both its function and timing. Functional verification may be used both at the formal and informal level. For example, during the debugging of a design, some informal tests may be applied to determine if the design meets its requirements. Functional verification is useful for signoff on a design.

2) Tradeoff Analysis: *During the design process, the system architect may use some form of testing as a means of evaluating tradeoffs [8].* Generally, these tradeoffs will be defined as an optimization problem.

3) Regression Testing: *This is used to ensure that a design change does not negatively effect the capabilities of the current design [8].* Regression testing combines the issues of completeness associated with signoff testing and the assumption that there will be changes in the model under test implied by tradeoff analysis.

2.4 Test Bench Generation System

The purpose of the test bench generation tool is to allow a signal processing system designer to configure a high-level test bench for VHDL models of a signal processing system. The overall structure of a test bench generation system is shown in Figure 2.2.

The *Test Bench Generation System* is a software system which allows the development of test benches including both the VHDL test bench components and the test data used to test the system [8]. The system developed here at Virginia Tech focuses on the construction of test benches at a high level of abstraction, thus minimizing the amount of detailed information that the modeler has to provide.

The components of the test bench generation system are:

- 1) Test bench model generator: It consists of a set of design tools that produce the VHDL model of the test bench.
- 2) Requirements capture tool: It consists of software tools like RDD-100 which help to interpret the specifications, as the test bench has to be an immediate interpretation of the specifications. Based on the system specification provided, the *secondary requirements* are derived from the *primary requirements*, and are both fed in as test bench generics.
- 3) Intelligent User Interface: Given all the VHDL components it helps to configure a test bench, generates data values for the test bench inputs based on user information obtained,

forms simulation control files for running the Synopsys simulator, forwards the output of the test bench to the Model Under Test, and stores the outputs of the test bench in data files for future use.

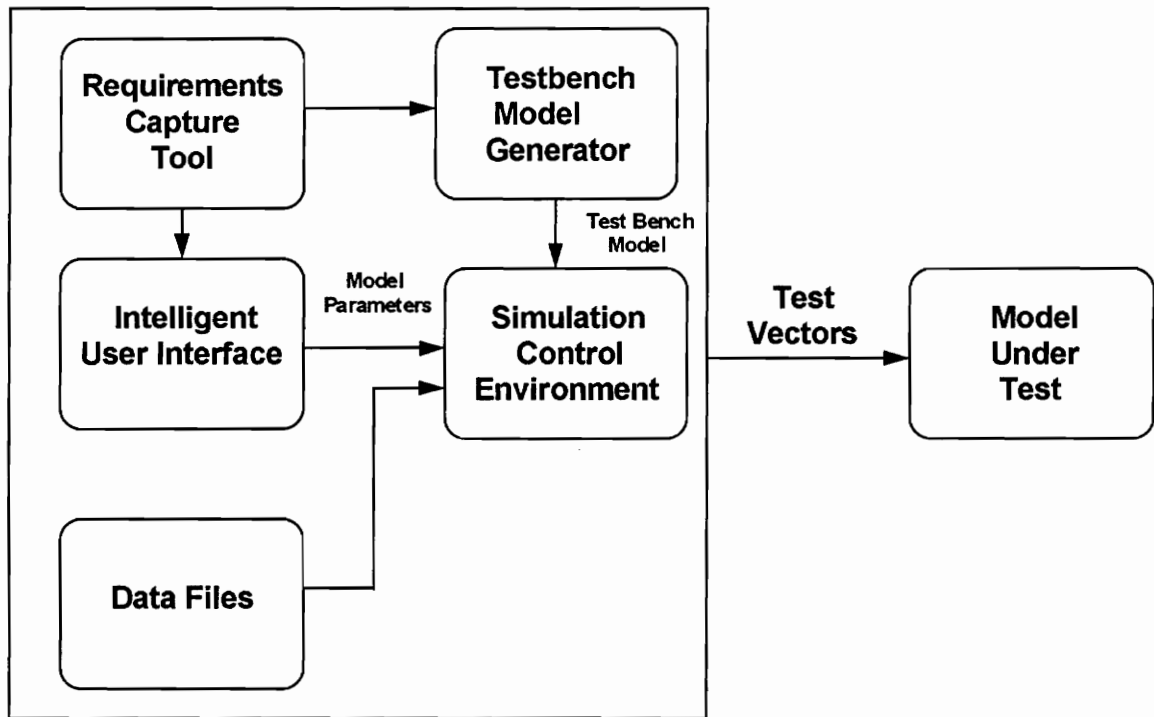


FIGURE 2.2 **Test Bench Generation System**

4) **Simulation Control Environment:** Interacts with the user through the Intelligent User Interface and the VHDL simulator while the test bench is being prepared for simulation, and during simulation. The VHDL simulator elaborates the test bench and performs the simulation.

Chapter 3. RASSP Specific Applications

3.1 Introduction

In this chapter, two of the RASSP applications for which testbenches have been developed are introduced. The first application is the Infra-Red Search and Track (IRST) algorithm [13]. The objective of the IRST algorithm is to detect moving targets in a sequence of image frames of an arbitrary size which may or may not have a moving background. The algorithm determines the best alignment of successive image frames by trying a combination of linear shifts of the first image frame in all directions within the scan range, subtracting the arrays of the first and second image frames for each shift, summing all the differences and then selecting the least total difference among all of the shifts tried. Based on the best alignment, the location and speed of the targets is determined.

The second application is the Synthetic Aperture Radar (SAR) [13-16]. SAR is an important tool for the collection of high-resolution, all-weather image data. The SAR

Chapter 3. RASSP Specific Applications

algorithm is used to provide a visual representation of the target and the surrounding environment in the form of a high resolution spatial map. The spatial map can be used for navigation and targeting and may also be recorded for post-flight analysis and intelligence gathering. This visual representation is generated based on the radar returns from the target.

3.2 IRST Testbench

This testbench has been modeled using the black box approach described in Chapter 2. The environment surrounding the IRST algorithm is modeled and given as input to the algorithm. The IRST testbench is thus responsible for providing the IRST algorithm with a sequence of image frames which contain both target and background (clutter) information merged together. The main component in the IRST testbench system is the stimulus generator which is responsible for producing the image frames.

3.2.1 IRST Stimulus Generator

Figure 3.1 shows the structure of the IRST stimulus generator. The components of the stimulus generator are: a target component, an environment component, a sensor component and a platform control component.

Target Component: The target component generates the locations of the target in X and Y co-ordinates, from the platform motion information and system parameters. In one mode of operation, the file I/O mode, the target component reads the locations of the targets directly from an external file and sends this information to the sensor component.

Chapter 3. RASSP Specific Applications

In another mode of operation, the user-interaction mode, the target component updates the locations of the targets every clock cycle, based on target parameters like initial location and velocity fed in by the user.

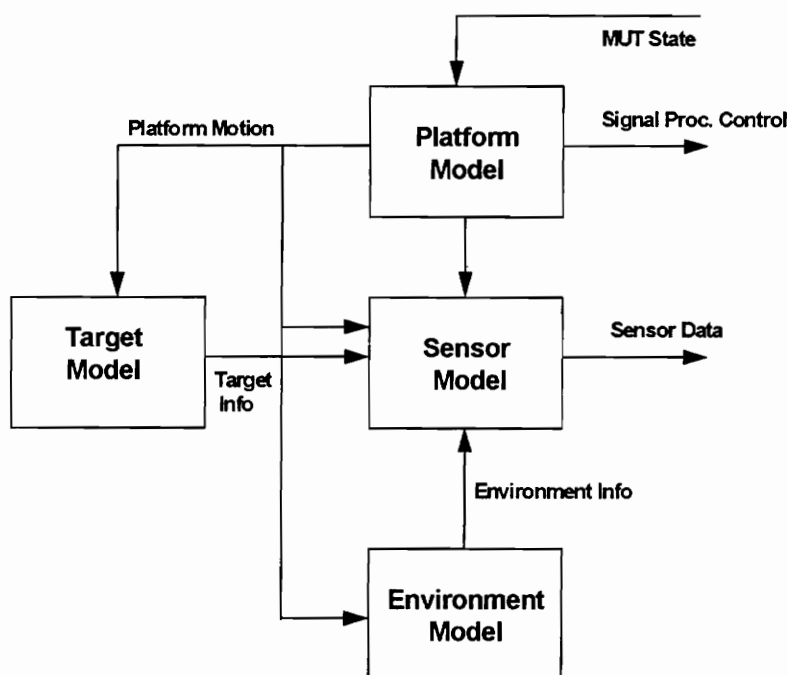


FIGURE 3.1 IRST Stimulus Generator

Several types of target motion like horizontal, vertical, climb and random motion are implemented in the target component. Based on the type of target motion, the locations of the targets in each successive frame are updated.

Chapter 3. RASSP Specific Applications

Environment Component: This component provides the clutter information to the sensor component. Clutter information could be mountains in the background, clouds or the horizon. Mountains and other such clutter are represented as an array of pixels corresponding to the actual shape. This array is forwarded to the sensor component. The environment component can also take information from the platform control component about the location and orientation of the platform while producing the clutter information. One way of generating clutter information is through data files which contain the clutter. In another method of clutter generation, the testbench model itself generates the clutter through a set of routines. When clutter information is stored in a data file, the user has a choice of several clutter files from which he/she can choose. Moving clutter is modeled by the environment component by reading in a sequence of clutter frames during successive frame updates. The sequence of clutter frames depict the moving clutter.

Sensor Component: The sensor component is the component where the target and clutter information (arrays) are merged together to produce the final image frame that will be fed to the MUT. Thus the sensor component generates the data that is actually seen by the signal processor. In a very simple testbench, the sensor component will just read data from a file and send it out to the MUT at the appropriate time. In an interactive testbench, the sensor component will generate the sensor data using inputs from the target component about the locations of the targets, input from the environment component about noise and clutter, and input from the platform control component about the location of the platform and the orientation of the sensor. The sensor update rate is an input parameter as is the number of frames to be produced. Thus the sensor component is responsible for producing the final output of the IRST testbench.

Platform control component : This component is used to provide control information for the MUT and platform location information for the sensor component. In this work the platform is assumed to be moving at a constant velocity. Also, in the IRST testbench implemented here, the sensor squint angle is 90 degrees perpendicular to the motion of the platform.

These four components form the IRST stimulus generator. The final output of the stimulus generator will be a sequence of image frames containing both target and clutter information.

3.3 SAR Testbench

The RASSP SAR Testbench [14] should have the capability to produce a sequence of digitized sensor data for the VHDL SAR image processor model. The SAR testbench implemented here is an interactive testbench - some of the input parameters are keyed-in by the user. The digitized sensor data are formed by several data-flow blocks embedded in the SAR sensor model.

Figure 3.2 shows the basic structure of the SAR testbench. The transmitted signal is a sequence of "chirp" pulses generated by frequency modulating a high-frequency RF carrier signal with a ramp signal. The received signal returned from the target is a complex signal and is basically the transmitted signal with a controlled delay. The main components of the SAR sensor model are the down converters, deramp section and A/D converter.

The inputs of the SAR sensor model are the technical parameters of the radar such as squint angle, carrier frequency, swath width, nominal range, pulse repetition frequency, chirp bandwidth and pulse width of the transmitted signal. Some of these parameters are used for producing the transmitted signal while other parameters are used in the various components.

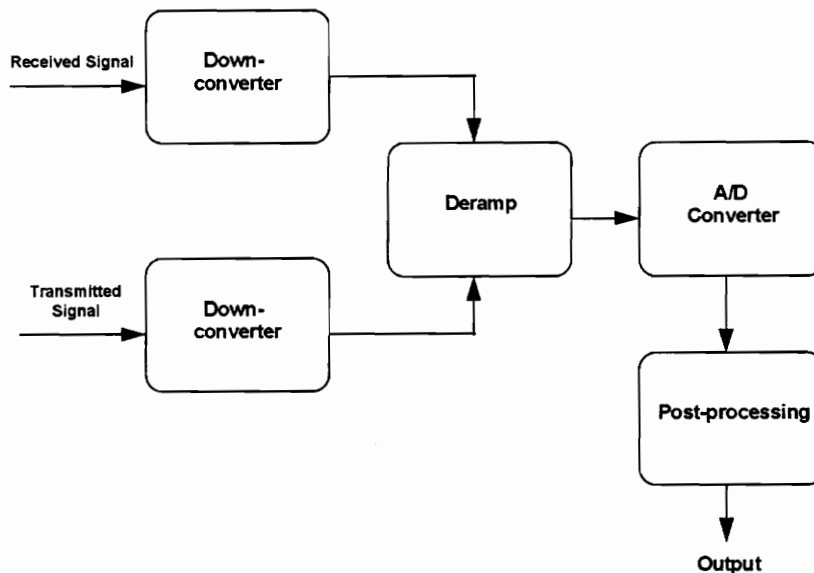


FIGURE 3.2 SAR Testbench

The functions of the various components in the SAR sensor model are:

Chapter 3. RASSP Specific Applications

Down-converter: The down-converter is used to remove the high-frequency carrier signal component from the transmitted and the received signals, thereby converting them to their baseband frequencies. The down-converted transmitted signal is also complex-conjugated.

Deramp section: The deramp section is basically a mixer whose two inputs are the complex-conjugate of the down-converted transmitted signal and the down-converted received signal. Thus the output of the deramp section is real and uncompressed.

A/D Converter and Post-processing: The real data is passed through the A/D converter where it is digitized. The post-processing block is responsible for gathering the output data and producing displays using MATLAB or any other similar tool.

Thus the output of the SAR sensor model implemented in this thesis is a sequence of samples that are produced by the A/D converter. The samples may be stored in a file for further analysis and also for forming 40-bit wide words which are fed as input to the SAR processor model.

Chapter 4. Requirements Capture

4.1 Introduction

Test bench development is based on the idea that the test bench has to be an immediate interpretation of the specification. The system specification consists of *General* requirements and *Specific* requirements which together specify the system being modeled.

General requirements specify the class of system being modeled. These requirements are the input to a high level design tool which generates VHDL code for the test bench. This is one of the inputs to the test bench generation system that is specified by the user. For example in the test bench generation system developed here at Virginia Tech, a typical general requirement is the choice between the applications in the application domain i.e. Infrared Search and Track application or the Synthetic Aperture Radar application. This general requirement is specified to the tool constructing the VHDL test bench code and to the Intelligent User Interface (IUI) which is used for user interaction with the test bench generation system, and for specifying the application area.

Specific requirements are characteristics of the application domain specified by the general requirements. The specific requirements are classified as *Primary* or *Secondary* requirements. Primary requirements are the independent parameters that are the input characteristics of the application domain chosen. Secondary requirements are dependent parameters derived from the primary requirements based on a math model specific to the application domain chosen. Both primary requirements and secondary requirements become values for test bench generics.

Figure 4.1 shown below depicts how the general and specific requirements fit into a high level test bench generation system.

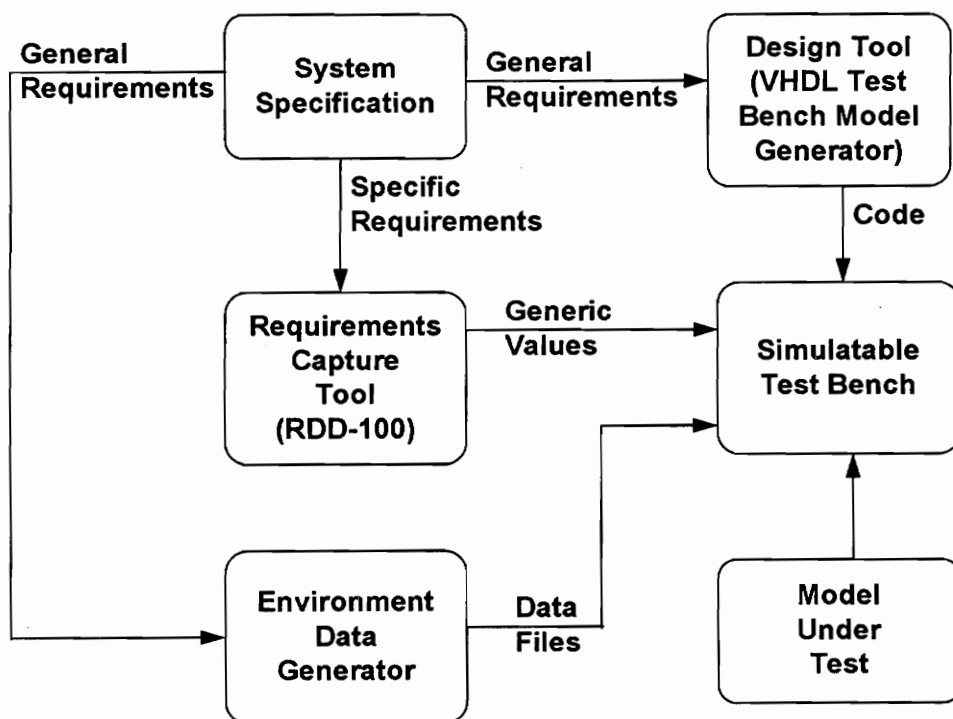


FIGURE 4.1 **A High Level Test Bench Generation System**

Chapter 4. Requirements Capture

The requirements capture tool used was RDD-100 - Requirements Driven Development [17]. The requirements capture tool uses a math model specific to the application domain to derive the secondary requirements from the primary requirements fed into the requirements capture tool.

This approach has been adopted to form a requirements interface for the Infrared Search and Track (IRST) application. This approach will be explained in the following section.

4.2 Requirements Interface for IRST

4.2.1 Primary and Secondary Requirements for IRST

Figure 4.2 depicts the RDD-100 requirements interface for IRST. The primary requirements for IRST were provided by the Research Triangle Institute (RTI). Figure 4.2 shows that the primary requirements for IRST application are:

Target Speed - Speed of the target specified in Machs.

Platform Type - The three different types of platforms possible are VF-X, VF, VP.

Sensor Resolution - The resolution of the sensor can be either High or Low.

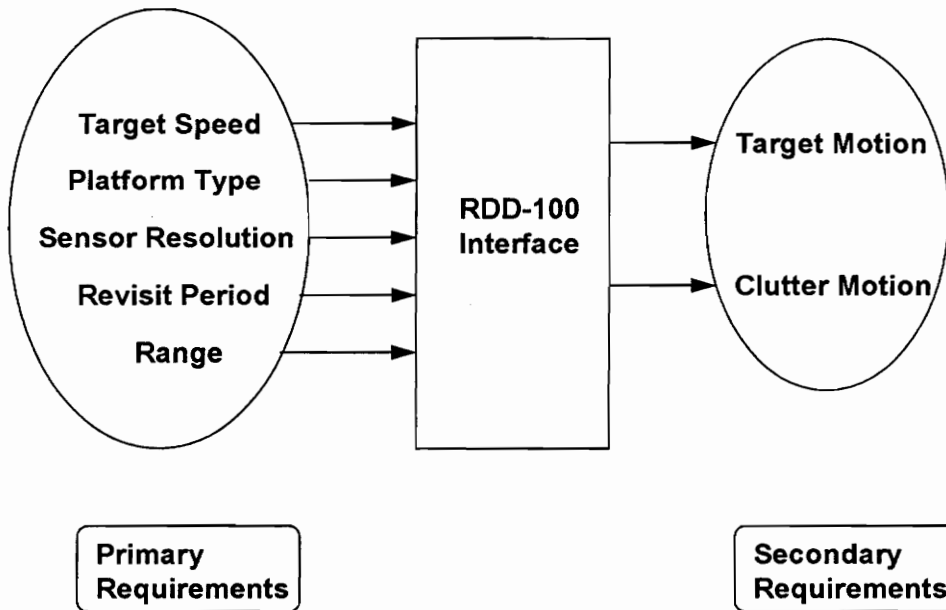


FIGURE 4.2 Requirements Interface for IRST

Revisit Period - This is the frame update rate, also fed in as a generic to the test bench.

Range - This indicates the range of the clutter and target from the platform.

The derived secondary requirements are:

Target Motion - This gives the movement of the target in pixels per frame derived based on the primary requirements.

$$\begin{aligned}\text{Total Angular Disp} &= a1 + a2 \\ \sin a1 &= \frac{(1/2 \times \text{PD})}{\text{Range 1}} \\ a1 + a2 &= 2 \times \text{asin}\left(\frac{\text{PD}}{2 \times \text{Range 1}}\right) \\ \text{Pixel Disp} &= \frac{(a1 + a2)}{\text{PAFOV}}\end{aligned}$$

FIGURE 4.3 Derivation of Formulae for Clutter and Target Motion

$$\begin{aligned}\text{Target Range} &= 1609.344 \times \text{Range} \\ \text{Target Velocity} &= \text{Mach to m/s} \times \text{Target speed} \\ \text{Target Disp} &= \text{Target Velocity} \times \text{Revisit Period} \\ \text{Target Motion} &= \frac{2 \times \text{asin}\left(\frac{\text{Target Disp}}{2 \times \text{Target Range}}\right)}{\text{Sensor Res Factor}}\end{aligned}$$

Figure 4.3 shows the derivation of the above formula used for calculating target motion

Typical values used for the primary requirements are Target Speed - 1.5 mach, Platform type VF-X, Sensor resolution - Low, Range - 200 miles, Revisit Period - 1 sec.

4.2.3 Realization of the Math Model using RDD-100

Definitions of terms related to RDD-100 [17]:

Behavior diagram is a graphical representation of the behavior exhibited by a function in terms of the time flow sequence, inputs and outputs.

Chapter 4. Requirements Capture

Function is a part of the system that carries out an action, usually converting an input to an output. For example, in a peach canning system there might be a function called Slice Peach, which accepts an Inspected Peach and produces Peach Slices [17].

Item is something that a function accepts or produces, in other words, an input or output. For example, a peach canning system has an input item called Inspected Peach and an output item called Peach Slices [17].

Discrete Function represents a function at the lowest level of detail of interest which is not broken down into sub-functions. This type of function accepts and produces only *Discrete Items*.

Time Function represents a higher level function that can have discrete functions and/or time functions as sub-functions. These sub-functions occur over a period of time and must be in their proper time sequence. A time function can have discrete items or *Time Items* as input or output.

The highest level time function developed was Requirements. The time function Requirements was decomposed into discrete functions - Initialize, Platform Data Init, Platform Data Proc, Clutter Data Proc, Target Data Proc 1, Target Data Proc 2, Target Data Proc 3. Figure 4.4 shows the decomposition of the time function Requirements. The inputs, outputs and the intermediate variables were stored as discrete items, and were declared to be global so that they are available to all the discrete functions.

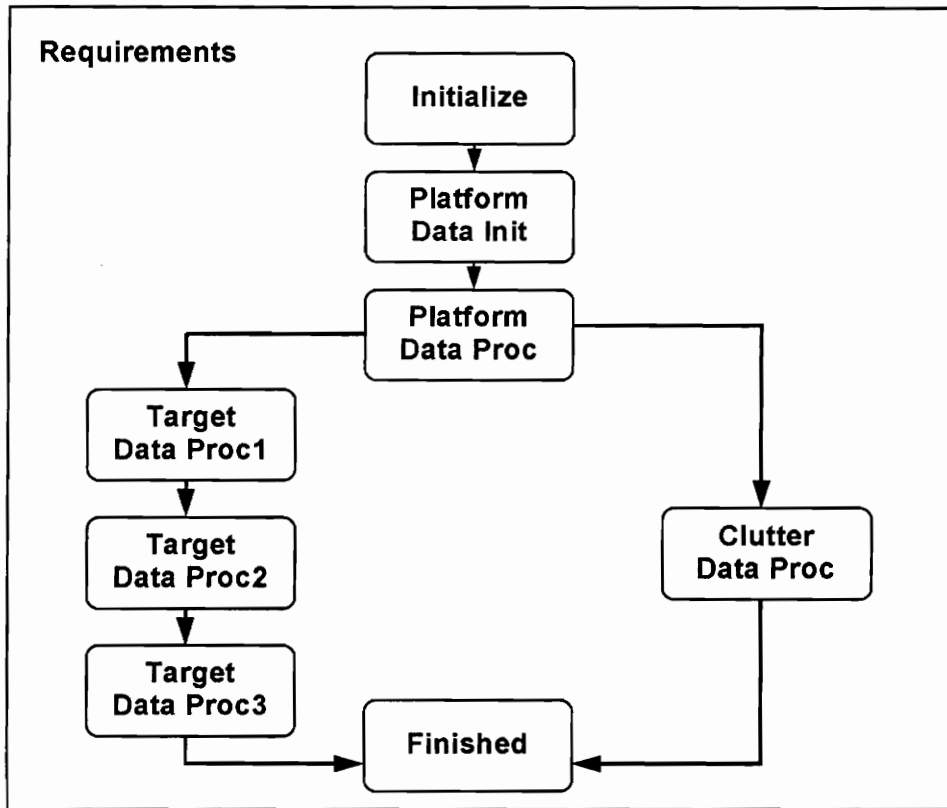


FIGURE 4.4 Time Function Requirements

Following is a brief description of each of the Discrete functions:

Initialize was used to assign values to the primary requirements Target Speed, Platform Type, Sensor Resolution, Range and Revisit Period, and based on the Sensor Resolution the Sensor Res factor was found.

Platform Data Init was used to initialize the conversion factor mach to m/s with 320 m/s, and based on the Platform Type the Platform Velocity was initialized.

Chapter 4. Requirements Capture

Platform Data Proc was used to calculate the platform displacement (Platform Disp) based on Platform Velocity and Revisit Period.

Clutter Data Proc was used to derive Clutter Range using Range, and based on the calculated Clutter Range the Clutter Motion was derived. This discrete function is executed concurrently with Target Data Proc 1.

Target Data Proc 1 was used to calculate Target Range using Range, and Target Velocity based on the mach to m/s conversion factor and the Target Speed in machs.

Target Data Proc 2 was used to calculate target displacement (Target Disp) based on Target Velocity and Revisit Period.

Target Data Proc 3 was used to calculate Target Motion based on the calculated Target Disp, Target Range and Sensor Res factor.

The time function Requirements was verified and simulated using the Dynamic Verification Facility provided in RDD-100. The primary inputs were assigned to the respective discrete items as a part of the Initialize discrete function, and the secondary outputs Target Motion and Clutter Motion were observed. The behavior diagram for time function Requirements is attached in Appendix A.

The event transcript file generated by RDD 100 during dynamic verification contains the values of the secondary requirements derived using the math model. This

Chapter 4. Requirements Capture

event transcript file can be parsed using a C program, and the values of the secondary requirements can be extracted and written to a file. As a part of the requirements interface, this file produced by the parser program containing the values of the secondary requirements can be used as input to the Intelligent User Interface (IUI).

Chapter 5. Library Primitives

5.1 Overview

A main emphasis of this thesis is the development of a library of primitives for different DSP based applications, so that structural test benches can be automatically generated by interconnecting these primitives together using a commercial schematic capture tool like Synopsys Graphical Environment (SGE) [18]. Based on the above approach library primitives have been developed for two DSP based applications - (1) Infrared Search and Track application and (2) Synthetic Aperture Radar. These applications have been explained in Chapter 3 on RASSP specific applications.

The library primitives developed have to support the model year concept. Model year design methodology is based on a successive refinement approach [7]. The current library of primitives is designed in accordance with the initial set of specifications. As a part of the successive refinement approach, the set of specifications could be upgraded to reflect recent technology or to correct functional errors present in the initial specification.

Chapter 5. Library Primitives

Since SGE is used to create the library of primitives, the upgraded specifications could be easily accommodated by correcting the related primitives to reflect the changes in the specifications. Thus by updating the related primitives, the test benches created in SGE as a structural interconnection of these primitives would also reflect the changes made in the specifications. The library primitives should also provide support for reuse. This is supported for by the use of SGE, as the individual primitives can be reused for other applications, and also the structure of the test bench could be reused for many similar applications, by changing the functionality of the components that are used in the test bench.

5.2 Features of Synopsys Graphical Environment (SGE)

The key features of Synopsys Graphical Environment (SGE), a commercial schematic capture tool are:

- An environment for creating and verifying designs.
- Supports structural model development.
- Design capture tools:
 - Symbol Editor - to create the library primitives.
 - Schematic Editor - to interconnect the library primitives based on the structure of the test bench model.
- VHDL interface to generate VHDL description of a design created using schematic capture.

5.3 Development of Library Primitives

Each of the library primitives are developed as a symbol in SGE, with the corresponding functionality being linked to it. The functionality of each of the primitives is written as associated VHDL code. The symbol for a primitive is created by specifying the input and output ports associated with the primitive. Once a symbol for a primitive is created the VHDL code for the symbol can be dumped by using the VHDL interface to SGE. The VHDL code dumped has the entity declaration part, and an empty architecture part where the VHDL code implementing the functionality of the primitive can be filled in. Some of the primitives can be developed by interconnecting precreated primitives to give a schematic. An associated symbol is created for the schematic. The VHDL code dumped for this symbol is a structural model consisting of component instantiations. The primitives for the Infrared Search and Track application and the Synthetic Aperture Radar application were developed using the above approaches.

5.3.1 Primitives for the Infrared Search and Track Test Bench

The stimulus generator for Infrared Search and Track application is composed of components - *osc*; the clock generator component, *environ*; the environment component, *target*; the target component and *sensor*; the sensor component. A brief description of each of these primitives is given below.

Osc :

This is the clock generator component whose clock period is based on the revisit period keyed in as a primary requirement.

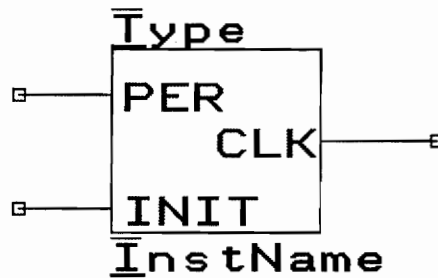


FIGURE 5.1 **SGE Symbol for Osc**

This is used to produce a clock pulse with period PER given by the revisit period, and is used to synchronize the other components when put together as a test bench. The oscillator starts producing clock pulses when the INIT signal goes to a '1', and keeps producing clock pulses as long as INIT is '1'. Figure 5.1 shows the symbol created in SGE for the oscillator component.

Environ:

This is the environment component which provides the clutter information to the sensor component. Clutter information could be mountains in the background, clouds or the horizon. Mountains and other such clutter are represented as an array of pixels corresponding to the actual shape. This array is forwarded to the sensor component. Thus, on every rising edge of CLK a clutter frame is produced. The clutter frames are 32x32 pixel arrays which are read from a file having a sequence of such clutter frames. Platform motion has been incorporated into this component. Different clutter frames are

read in every clock cycle thus depicting a simple case of platform motion. Figure 5.2 shows the symbol created in SGE for the environment component.

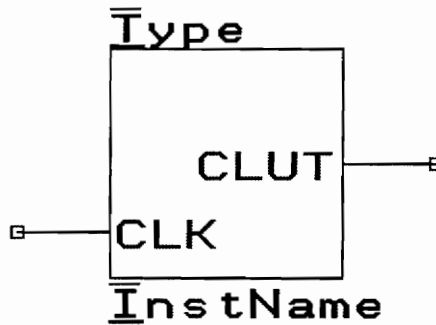


FIGURE 5.2 **SGE Symbol for Environ**

Target :

Target Component : The target component generates the locations of the target in X and Y co-ordinates, from the platform motion information and system parameters. This component produces a 32x32 pixel array which has zeros in all the location except for a 255 in the location (X,Y) where (X, Y) is the location of the target in the current frame. The target updates are done at every clock cycle when RUN is '1'. When INIT is '1' and RUN is '0' there are no updates in targets made at every clock cycle. In one mode of operation, the file I/O mode, i.e. when MODE is '1' the target component reads the locations of the targets directly from an external file at every clock cycle.

In another mode of operation, the user-interaction mode, i.e. when MODE is '0' the target component updates the locations of the targets every clock cycle, based on target parameters like initial location, velocity, and angle fed in by the user.

Chapter 5. Library Primitives

MODE_MOTION is used to define the movement of the target, i.e. where 1 means horizontal motion, 2 - vertical motion, 3 - climb and 4 - random walk. TARG_RANGE is an input based on the distance of the target from the platform and is used to hide targets based on the region of clutter it is in and the clutter range in that region, which are defined by generic values. When target range is more than the clutter range the target is hidden. Hiding is implemented by not updating FRAME during the respective clock cycle. When target range is less than the clutter range the target is visible. Figure 5.3 shows the symbol created in SGE for the target component.

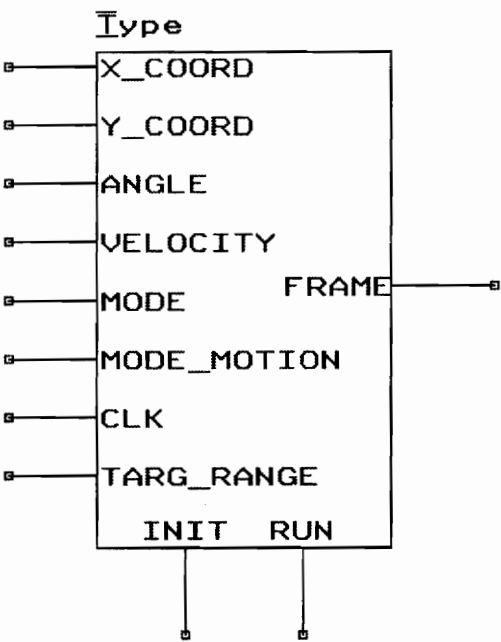


FIGURE 5.3 SGE Symbol for Target

Sensor :

The sensor component is the component where the target and clutter information (arrays) are merged together to produce the final image frame that will be fed to the Model Under

Test (MUT). This component adds the 32x32 clutter array with the 32x32 frame array containing the target information to produce the final image array. This operation is done at every clock cycle as long as INIT is '1'. Thus an image frame is produced every clock cycle, which is equivalent to the frame update rate. Figure 5.4 shows the symbol created in SGE for the sensor component.

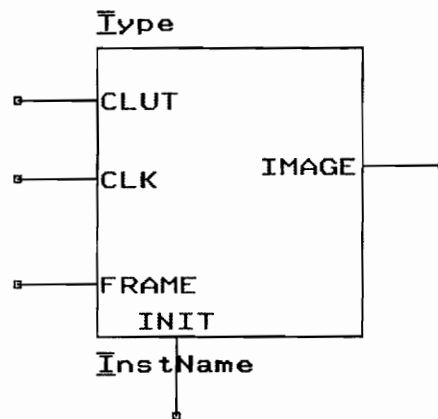


FIGURE 5.4 SGE Symbol for Sensor

5.3.2 Primitives for the Synthetic Aperture Radar Test Bench

The stimulus generator for the Synthetic Aperture Radar application is composed of components: chirp , comptfp, compmul, genchirp, delay, neg, pass, decimate and change. The genchirp primitive is composed of chirp, comptfp and compmul. A brief description of each of these primitives is given below.

Chirp :

This block is used to generate the chirp signal, which in addition to a complex tone is used to form the transmitted signal from the radar. The inputs to the primitive are the chirp

bandwidth (BW), sampling frequency (SAMP_FREQ), pulse width (PW), pulse repetition frequency (PRF), number of samples (N). The outputs are the real part (R) and the imaginary part (I) of the chirp signal generated. Figure 5.5 shows the symbol created in SGE for the chirp component.

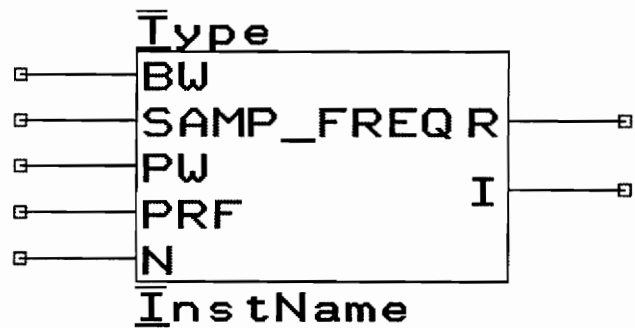


FIGURE 5.5 SGE Symbol for Chirp

Comptfp :

This block is used to produce the complex tone, which is used as a carrier for the chirp signal to be transmitted from the radar. The inputs are the carrier frequency (FREQ), sampling frequency (SAMP_FREQ) and number of samples (N). The outputs are real part (R) and the imaginary part (I) of the generated complex tone. Figure 5.6 shows the symbol created in SGE for the comptfp component.

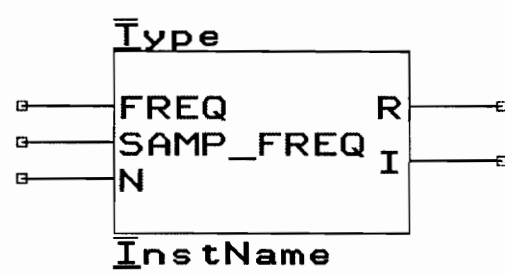


FIGURE 5.6 SGE Symbol for Comptfp

Compmul :

This block is used as a complex multiplier. Figure 5.7 shows the symbol created in SGE for the compmul component. Two complex inputs are fed in in terms of their respective real and imaginary parts RIN1, RIN2 and IIN1, IIN2. The resultant complex output is fed out in terms of their real and imaginary parts ROUT and IOUT.

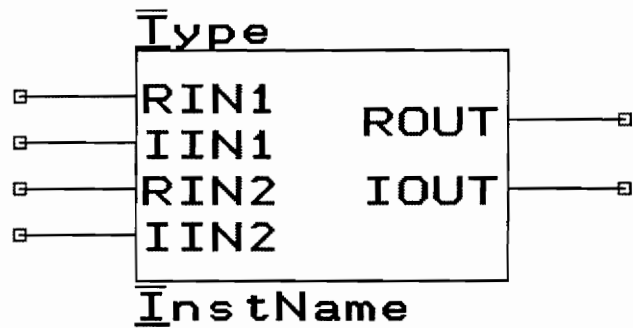


FIGURE 5.7 SGE Symbol for Compmul

Genchirp :

This block is used to generate the transmitted signal from the radar by modulating the chirp signal with the carrier signal. Figure 5.8 shows the symbol created in SGE for the genchirp component.

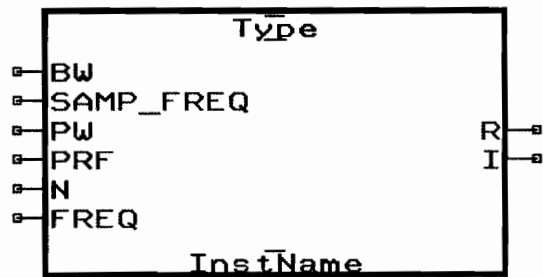


FIGURE 5.8 SGE Symbol for Genchirp

Figure 5.9 shows the schematic created in SGE for genchirp which depicts the decomposition of the genchirp block in terms of the chirp, comptfp and compmul blocks.

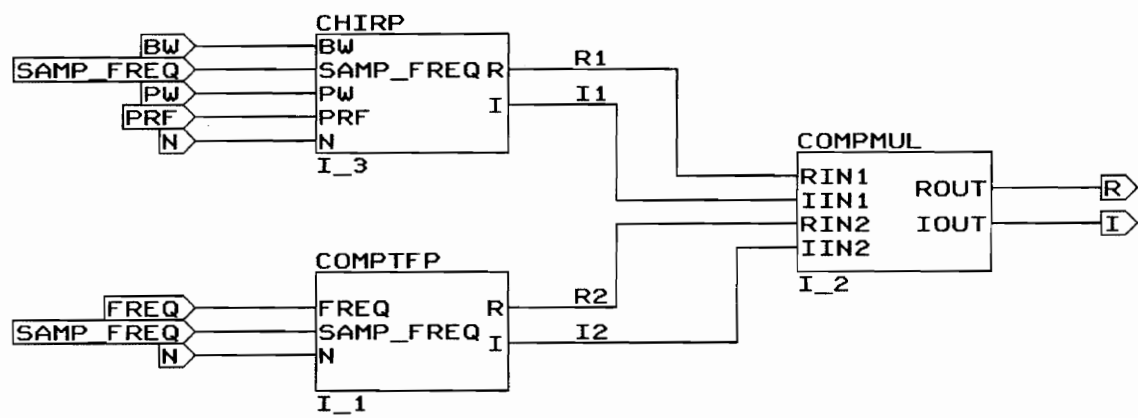


FIGURE 5.9 SGE Schematic for Genchirp

Delay :

This block is used to simulate the received signal by producing a delayed version of the transmitted signal. This delay introduced, is specified by a generic and is proportional to the distance of the target region from the radar. The generic DEL for delay is not displayed as a part of the symbol, but could be depicted using the text option in SGE. Figure 5.10 shows the symbol created in SGE for the delay component.

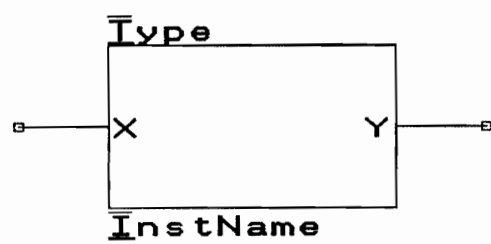


FIGURE 5.10 SGE Symbol for Delay

Neg :

This block is used to compute the complex conjugate. The inputs are fed in as real part RX and imaginary part IX. The computed complex conjugate is fed out as real part RY and complex part IY. Figure 5.11 shows the symbol created in SGE for neg.

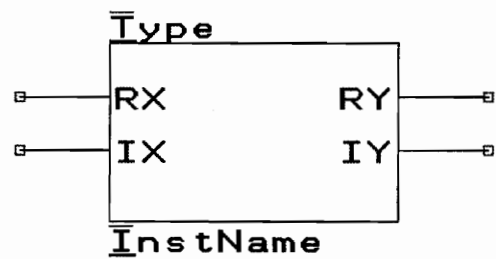


FIGURE 5.11 SGE Symbol for Neg

Decimate :

This block is used to resample data using decimation, there by reducing the number of samples. The decimation factor DEC is fed in as a generic. The inputs are the real and imaginary parts of the signal RX and IX, and the number of samples N. The outputs are the real and imaginary parts of the output sequence. Figure 5.12 shows the symbol created in SGE for the decimate component.

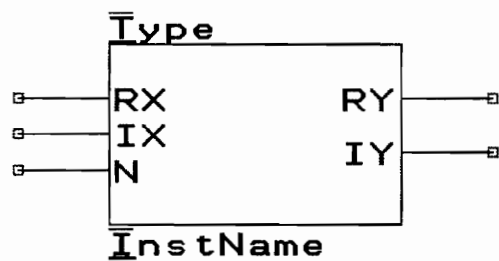


FIGURE 5.12 SGE Symbol for Decimate

Change :

This block is used to change the data format from real to bit vector. The input is RIN which is real and the output is BOUT which is in bit vector format. Figure 5.13 shows the symbol created in SGE for the change component.

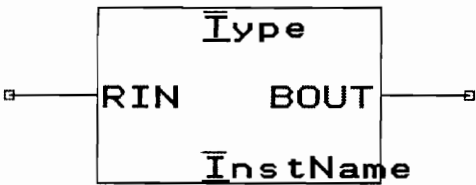


FIGURE 5.13 SGE Symbol for Change

Pass :

This component just passes the input to the output, and is used to make compensations for delta delay problems during simulation. For example in the overall SAR test bench explained in Chapter 6, the PASS block is used to compensate for the delta delay introduced by the COMPMUL block. Thus by introducing the PASS block the outputs of the down converting section are synchronized. Figure 5.14 shows the symbol created in SGE for the pass component.

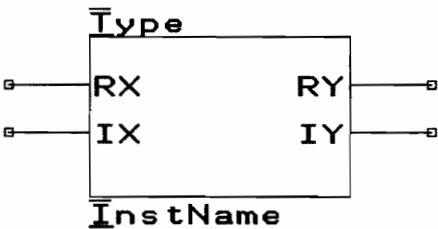


FIGURE 5.14 SGE Symbol for Pass

5.4 Parameterization of the Primitives

When modeling digital systems, the desired level of abstraction often dictates the set of primitives that will be used in the model. If modeling is done at the gate level, then the primitive set will consist of devices such as inverters, gates, and flip-flops. At a higher level of abstraction, the primitives might include counters, registers, multiplexers and decoders. At an even higher level, purely functional primitives can be used (i.e. primitives that don't have a direct correspondence to actual hardware elements). For example, primitives such as FETCH, EXECUTE, and INTERRUPT could be used to model traditional von Neumann type computer architectures. With VHDL, it is possible to create models using the above primitive set.

Since, for a given level of abstraction, if the set of primitives is fairly well-defined, it is plausible to expect a significant decrease in the time required to develop a model if the modeler has access to a pre-coded library of primitives.

The basic primitive design philosophy can be stated as follows:

If the modeler is provided with a complete set of error-free primitives for the desired level of abstraction, then these primitives can be used to create an error-free description of the required model.

The above philosophy can be applied to structural test bench development using a library of primitives, from which an error free structural test bench can be created by interconnecting the appropriate primitives from the library.

Chapter 5. Library Primitives

The two main features that the primitives developed as a part of the library of primitives of the test bench should have are:

- 1) The primitives should support reuse, so that primitives developed for a particular application could be used for other similar applications.
- 2) The primitives should support the model year concept, as this is a main emphasis of the RASSP test bench generation task. The model year concept is based on successive refinement approach [7], where the features of the model under test are improved form year to year. The main requirements of the model year concept are:

- Test bench development should be rapid.
- Test bench modules should allow for easy modifications to reflect the model year changes in the module under test.
- The granularity of sub modules in the test bench should change to reflect varying levels of abstraction in the design.
- Use of realistic data files as inputs to the test bench model.

The primitives for the Infrared Search and Track (IRST) application and the Synthetic Aperture Radar (SAR) application explained in section 5.3 satisfy all these requirements. The main approach used in parameterization of the primitives, is to make the primitives as general as possible by the use of generic values and appropriate input ports. For example in the TARGET module, a primitive in the Infrared Search and Track application the different generics used are: the ranges of the different clutter regions REG1_RANGE, REG2_RANGE, REG3_RANGE and the size of the two dimensional array FRAME produced as an output, which is equal in size to the final image frame produced by the IRST test bench. VHDL code for the TARGET module is shown in Figure 5.15.

Chapter 5. Library Primitives

```
-- VHDL Model Created from SGE Symbol target22.sym -- Nov 30 00:06:00 1994

use work.all;
use STD.TEXTIO.all;
use work.USER_TYPES1.all;
use work.MATH_REAL.all;

entity TARGET22 is
    Generic ( DEL_TIME:= 0 NS;
              REG1_RANGE : INTEGER := 5;
              REG2_RANGE : INTEGER := 10;
              REG3_RANGE: INTEGER := 15;
              FRAME_SIZE : INTEGER := 32 );
    Port ( ANGLE : In  INTEGER;
           CLK : In  BIT;
           INIT : In  BIT;
           MODE : In  BIT := '0';
           MODE_MOTION : In  INTEGER;
           RUN : In  BIT;
           TARG_RANGE : In  INTEGER;
           VELOCITY : In  INTEGER;
           X_COORD : InOut INTEGER;
           Y_COORD : InOut INTEGER;
           FRAME : Out  INT_2D(0 TO 31,0 TO 31) := (OTHERS => (OTHERS
                                                                    => 0)) );
end TARGET22;

architecture BEHAVIORAL of TARGET22 is

begin

    process(INIT,CLK,RUN)

        variable FRAME_TEMP: INT_2D(0 to 31,0 to 31):= (others => (others => 0));
        variable VLINE1: LINE;
        variable ANGLE_RAD: REAL;
        variable TEMP_ANGLE: INTEGER;
        variable I: INTEGER:= 0;
        variable I1: INTEGER:= 1;
        variable J: INTEGER:= 0;
        variable NEW_X_COORD,NEW_Y_COORD:INTEGER:= 0;
```

Chapter 5. Library Primitives

```
variable TEMP1: INTEGER:= 0;
variable TEMP_X,TEMP_Y: INTEGER := 0;
variable TAR_POS: INT_2D(0 to 20,0 to 1):= (others => (others => 0));
variable FLAG1 : BIT := '0';
variable START_FLAG : BIT := '0';
variable RAND_NO1,RAND_NO2 : REAL;
variable SEED1: INTEGER := 2000;
variable SEED2: INTEGER := 5000;
file POSITION : TEXT is in
    "/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/target.in";

begin

if MODE = '0' then

    if INIT = '1' and RUN = '0' then
        NEW_X_COORD := X_COORD;
        NEW_Y_COORD := Y_COORD;
    end if;

    if RUN = '1' then
        if CLK'EVENT and CLK = '1' then

            FRAME_TEMP(TEMP_X,TEMP_Y) := 0;

            case MODE_MOTION is
                when 1 => TEMP_ANGLE := 0;
                    FLAG1 := '0';
                when 2 => TEMP_ANGLE := 90;
                    FLAG1 := '0';
                when 3 => TEMP_ANGLE := ANGLE;
                    FLAG1 := '0';
                when 4 =>
                    UNIFORM(SEED1,SEED1,RAND_NO1);
                    UNIFORM(SEED2,SEED2,RAND_NO2);
                    FLAG1 := '1';
                when others => null;

            end case;
```

```
if ((FLAG1 = '0') and (START_FLAG = '1')) then
    ANGLE_RAD := MATH_DEG_TO_RAD*REAL(TEMP_ANGLE);
    NEW_X_COORD := NEW_X_COORD +
        INTEGER(REAL(VELOCITY)*COS(ANGLE_RAD));
    NEW_Y_COORD := NEW_Y_COORD +
        INTEGER(REAL(VELOCITY)*SIN(ANGLE_RAD));
elsif ((FLAG1 = '1') and (START_FLAG = '1')) then
    NEW_X_COORD := INTEGER(31.0*RAND_NO1);
    NEW_Y_COORD := INTEGER(31.0*RAND_NO2);
end if;

if ((NEW_X_COORD<FRAME_SIZE) and (NEW_Y_COORD<FRAME_SIZE))
then

    X_COORD <= NEW_X_COORD;
    Y_COORD <= NEW_Y_COORD;

    TEMP_X := 31-NEW_Y_COORD;
    TEMP_Y := NEW_X_COORD;

    if (15*NEW_Y_COORD <= (-31*NEW_X_COORD+465)) then
        if REG1_RANGE > TARG_RANGE then
            FRAME_TEMP(TEMP_X,TEMP_Y) := 255 ;
        end if;
    elsif ((7*NEW_Y_COORD <= (5*NEW_X_COORD+20)) and
        (15*NEW_Y_COORD > (-31*NEW_X_COORD+465))) then
        if REG2_RANGE > TARG_RANGE then
            FRAME_TEMP(TEMP_X,TEMP_Y) := 255 ;
        end if;
    else
        if REG3_RANGE > TARG_RANGE then
            FRAME_TEMP(TEMP_X,TEMP_Y) := 255 ;
        end if;
    end if;

    FRAME <= FRAME_TEMP after DEL;
```

```
        START_FLAG := '1';

        end if;
    end if;
end if;

elsif MODE = '1' then

    if INIT = '1' and RUN = '0' then

        while not(ENDFILE(POSITION)) loop

            READLINE(POSITION,VLINE1);

            for J in 0 to 1 loop
                READ(VLINE1, TEMP1);
                TAR_POS(I,J):= TEMP1;
            end loop;

            I:= I+1;

        end loop;

        NEW_X_COORD := TAR_POS(0,0);
        NEW_Y_COORD := TAR_POS(0,1);

    end if;

    if RUN = '1' then
        if CLK'EVENT and CLK = '1' then

            FRAME_TEMP(TEMP_X,TEMP_Y) := 0;

            if ((NEW_X_COORD<FRAME_SIZE) and
                (NEW_Y_COORD<FRAME_SIZE)) then

                X_COORD <= NEW_X_COORD;
                Y_COORD <= NEW_Y_COORD;
```

```
TEMP_X:= 31-NEW_Y_COORD;
TEMP_Y := NEW_X_COORD;

if (15*NEW_Y_COORD <= (-31*NEW_X_COORD+465)) then
    if REG1_RANGE > TARG_RANGE then
        FRAME_TEMP(TEMP_X,TEMP_Y) := 255 ;
    end if;
elsif ((7*NEW_Y_COORD <= (5*NEW_X_COORD+20)) and
(15*NEW_Y_COORD > (-31*NEW_X_COORD+465))) then
    if REG2_RANGE > TARG_RANGE then
        FRAME_TEMP(TEMP_X,TEMP_Y) := 255 ;
    end if;
else
    if REG3_RANGE > TARG_RANGE then
        FRAME_TEMP(TEMP_X,TEMP_Y) := 255 ;
    end if;
end if;

FRAME <= FRAME_TEMP after DEL;

end if;

if not((TAR_POS(I1,0)=0) and (TAR_POS(I1,1)=0)) then
    NEW_X_COORD := TAR_POS(I1,0);
    NEW_Y_COORD := TAR_POS(I1,1);
    I1 := I1+1;
end if;

end if;

end if;

end process;

end BEHAVIORAL;
```

```
configuration CFG_TARGET22_BEHAVIORAL of TARGET22 is
  for BEHAVIORAL
    end for;
end CFG_TARGET22_BEHAVIORAL;
```

FIGURE 5.15 VHDL Code for TARGET Module

The use of input ports like MODE in the TARGET module help to make the TARGET module more general by incorporating both the file I/O and the user interaction mode in the same model. Other input ports that make the TARGET module general are: MODE_MOTION and TARG_RANGE. MODE_MOTION helps to incorporate different target movements onto the same model, i.e. , where MODE_MOTION is 1 - horizontal; 2 - vertical; 3 - climb and 4 - random. TARG_RANGE helps to produce output array FRAME with hidden targets based on the values of TARG_RANGE, REG1_RANGE, REG2_RANGE and REG3_RANGE. Similarly the use of generics for primitives of the SAR application like generic DEL, the delay introduced in the DELAY block, generic DEC, the decimation factor used in the DECIMATE block make the primitives more general.

Chapter 6. Structural Model Construction

6.1 Methodology

The approach used in structural test bench development is to create schematics interconnecting the appropriate primitives from the library of primitives using a commercial schematic capture tool like Synopsys Graphical Environment (SGE) [18]. The schematic editor facility is used to interconnect the symbols created in the SGE symbol editor corresponding to the different primitives. The ports for the overall test bench are specified as a part of the schematic. A symbol is created corresponding to the schematic developed for the test bench. The VHDL interface is used to dump the VHDL code from the symbol created for the overall test bench. The VHDL code dumped for the structural test bench consists of component instantiations corresponding to the different primitives used in the schematic with port maps corresponding to the interconnections between the primitives. The resultant VHDL code for the test bench developed is then simulated using the Synopsys VHDL simulator.

6.2 Structural Test Bench Generation

Based on the approach explained in section 6.1 structural test benches were developed for the Infrared Search and Track application and the Synthetic Aperture Radar application. The test benches developed for these two applications were composed of the primitives explained in chapter 5. VHDL code for the test benches are attached in Appendix B.

6.2.1 Infrared Search and Track Test Bench

The test bench for the Infrared Search and Track application is composed of components - *osc*; the clock generator component, *environ*; the environment component, *target*; the target component and *sensor*; the sensor component. A brief description of each of these primitives is given in chapter 5. Figure 6.1 shows the symbol created in SGE for the IRST test bench and figure 6.2 shows the schematic created in SGE for the IRST test bench. The symbol was created from the schematic.

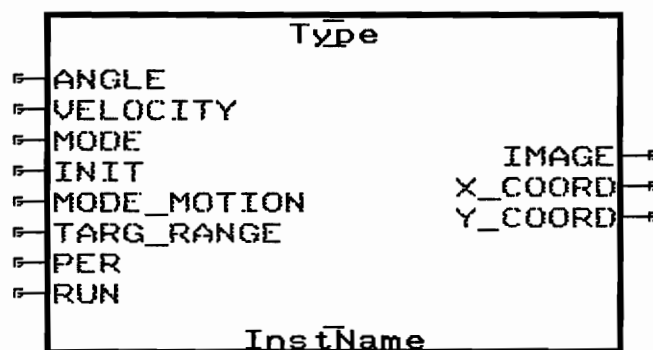


FIGURE 6.1 SGE Symbol for IRST Testbench

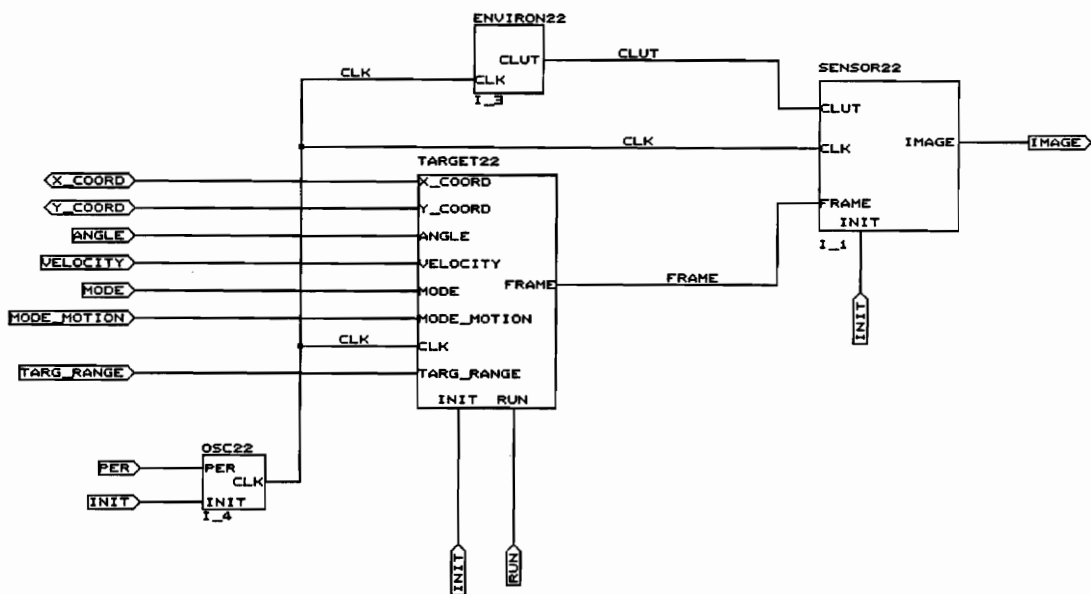


FIGURE 6.2 SGE Schematic for IRST Testbench

The basic function of the IRST test bench is to produce 32x32 image frames every clock cycle, as long as the INIT signal is high. The image frames have both clutter and target information. The oscillator component produces a clock with period corresponding to the revisit period or the frame update rate specified. The clutter information is provided by the environment component which produces 32x32 clutter frames every clock cycle. These clutter frames are read from a file which has a sequence of clutter frames corresponding to the different image frames. The target information is provided by the target component which produces a 32x32 array called Frame which contains a value of 255 at the location given by the calculated X_COORD and Y_COORD. The values of X_COORD and Y_COORD are updated every clock cycle as long as both INIT and RUN are high. Based on the mode of operation, the X_COORD and Y_COORD are calculated differently. In the file I/O mode these values are read directly from a file containing target

information corresponding to the different image frames. In the user interaction mode the X_COORD and Y_COORD values are updated every clock cycle based on all the relevant inputs. The sensor component produces the final image frame fed to the Model Under Test, by adding the clutter array produced by the environment component and the frame array produced by the target component. The image frames are produced every clock cycle by the sensor component, as long as INIT signal is high. Thus the IRST test bench produce an image frame every clock cycle as long as INIT signal is high. When INIT is high and RUN is low, the image frame only contains clutter information. When INIT is high and RUN is high the image frame contains both clutter and target information.

6.2.2 Synthetic Aperture Radar Test Bench

The stimulus generator for the Synthetic Aperture Radar (SAR) application is composed of components: chirp , comptfp, compmul, genchirp, delay, neg, pass, decimate and change.

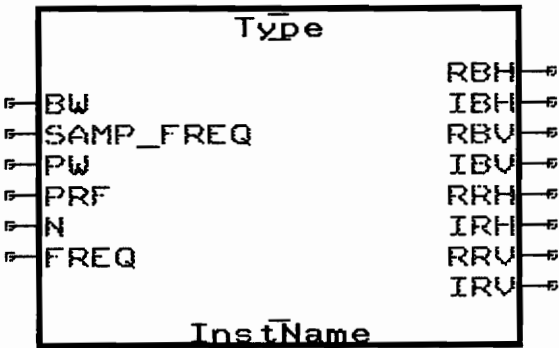


FIGURE 6.3 SGE Symbol for SAR Testbench

Chapter 6. Structural Model Construction

The genchirp primitive is composed of chirp, comptfp and compmul. Figure 6.3 shows the symbol created in SGE for the SAR test bench and figure 6.4 shows the schematic created in SGE for the SAR test bench. The symbol was created from the schematic. The inputs to the SAR test bench are the chirp bandwidth (BW), the sampling frequency (SAMP_FREQ), the pulse width (PW), the pulse repetition frequency (PRF), the number of samples (N) and the carrier frequency (FREQ).

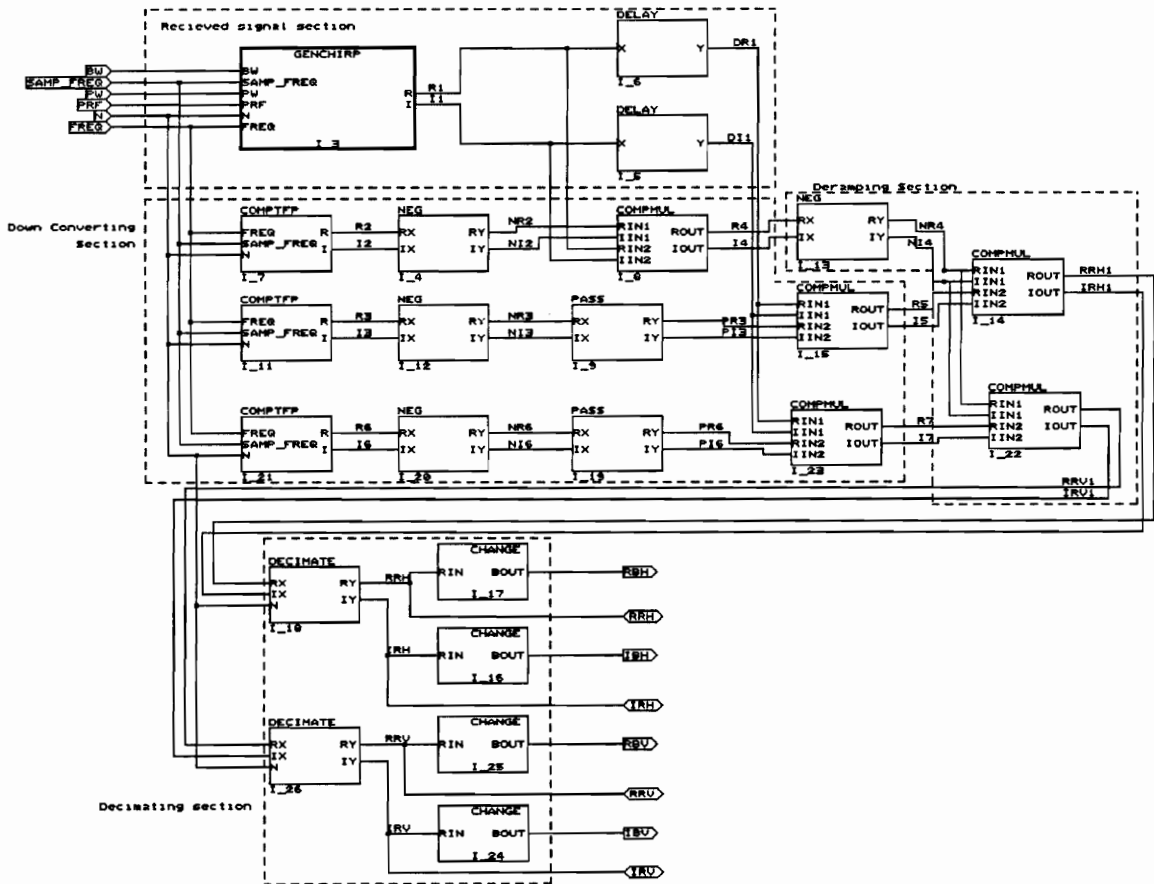


FIGURE 6.4

SGE Schematic for SAR Testbench

Chapter 6. Structural Model Construction

The outputs of the SAR test bench are the real and imaginary parts of the horizontal component in bit vector format (RBH and IBH), real and imaginary parts of the horizontal component as real numbers (RRH and IRH), the real and imaginary parts of the vertical component in bit vector format (RBV and IBV), the real and imaginary parts of the vertical component as real numbers (RRV and IRV).

As shown in the schematic above the SAR test bench can be split into four sub sections. The sections are: 1) Received signal section 2) Down converting section 3) Deramping section 4) Decimating section. The overall schematic for the SAR test bench was organized based on these sub sections. The blocks used to construct these sub sections are explained in Chapter 5 under the title Primitives for the SAR test bench. A brief description of the sub sections is given below.

1) Received signal section :

This section is used to generate the received signal by delaying the transmitted signal generated. The transmitted signal from the radar is generated using the Genchirp block. Both the real and imaginary parts of the transmitted signal generated are delayed by using the Delay block.

2) Down converting section :

In this section both the transmitted signal and the received signal are down converted to the base band frequencies by demodulating the transmitted and the received signal with the complex tone used as the carrier in these signals. The Comptfp block is used to generate the complex tone used as a carrier in the transmitted and the received signal. The transmitted and received signal can be down converted by multiplying these signals with

Chapter 6. Structural Model Construction

the complex conjugate of the complex tone. The Neg block is used to generate the complex conjugate of the complex tone. The Compmul block is used to multiply the transmitted and received signal with the complex conjugate of the complex tone. The resultant signals are the transmitted and received signal down converted to the base band frequencies respectively.

3) Deramping section :

In this section the down converted versions of the transmitted and received signal are correlated together, i.e., deramped. The complex conjugate of the down converted version of the transmitted signal is generated using the Neg block. This signal is multiplied with the down converted version of the received signal using the Compmul block. Two of these compmul blocks are used to simulate the horizontal and the vertical components.

4) Decimating section :

In this section the outputs of the Deramping section are decimated using the Decimate block so as to reduce the number samples in the output based on the decimation factor. The real and imaginary parts of the decimated signals are then converted from real to bit vector format using the Change block.

Thus the output of the SAR test bench model is a sequence of samples that are produced by the A/D converter (Change block). The samples may be stored in a file for

further analysis and also for forming a sequence of 40-bit wide words which are fed as input to the SAR processor model.

6.3 Procedure based Test Bench Model for IRST

Package VER3V_PRIM1 consisting of the procedure equivalents of the primitives osc, environ, target and sensor for the IRST test bench explained in Chapter 5 was also created. The procedure equivalents for the primitives were created by developing variable based procedures which modeled the functionality of the primitives. A variable based test bench for IRST was modeled as a sequence of procedure calls. The package VER3V_PRIM1 and the variable based test bench for IRST have been attached in Appendix B.

Chapter 7. Intelligent User Interface

7.1 Purpose

The Intelligent User Interface (IUI) is one of the main components of the test bench generation system explained in Chapter 2. Given all the VHDL components that are to be used in the test bench, the primary function of the IUI is to configure a test bench. The IUI was fully developed in C on a SUN-OS platform. A code listing is attached in Appendix C.

Functions of the Intelligent User Interface (IUI) are :

- 1) To apply the system requirements derived using the requirements capture tool and from other sources to the test bench model.
- 2) To control the values of generics and to feed in values for the input ports in the test bench model, e.g., target initial position, velocity, angle and mode of motion in the case of Infrared Search and Track application. This information is derived from the requirements

Chapter 7. Intelligent User Interface

capture tool or through user interaction. The test bench models use signals to hold generic values because signals can be changed on line during simulation, while VHDL generics remain constant during simulation.

- 3) To form the simulation control file which is used to run the Synopsys simulator.
- 4) To control the sequencing of simulation and display of the results based on information input through the IUI.
- 5) Generation of the data values for the test bench inputs.
- 6) To act as an interface between the test bench model and the Model Under Test, as it helps to forward the output of the test bench to the Model Under Test, and also to store the output in data files for future use.

Thus the IUI helps to integrate the test bench model produced by the test bench generator into one system.

7.2 Menu Structure

The IUI provides the user with a set of menus which helps the user to configure the test bench model and to provide inputs to the test bench model. The IUI either prompts the user with a list of possible selections, or asks the user to key in input values within a certain range. Figure 7.1 shows the overall menu structure provided by the IUI, developed here at Virginia Tech.

The set of menus provided as a part of the IUI are :

1) User/Requirement Driven :

This menu gives the option of running the test bench with input values specified using the requirements capture tool or running the test bench with user driven choices. The requirements driven case has been developed for the Infrared Search and Track (IRST) application, and in this case the primary requirements are fed into the requirements capture tool from a file, which then produces the secondary requirements that are fed as inputs to the test bench.

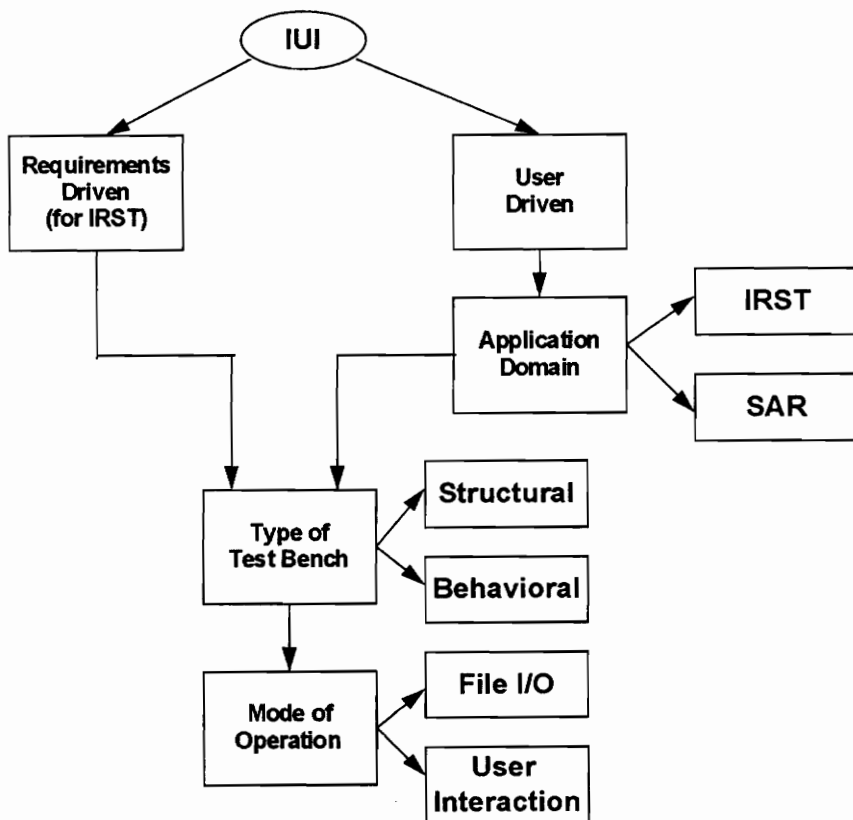


FIGURE 7.1

Menu Structure in IUI

Chapter 7. Intelligent User Interface

2) Application domain :

This menu allows the user to choose from the two applications - Infrared Search and Track (IRST) application or the Synthetic Aperture Radar (SAR) application. This choice is requested if the user driven option was chosen as the previous menu option. Based on this choice the respective test bench is generated by the test bench generator, and other menu choices are selected.

3) Type of Test Bench :

This menu allows the user to choose between a structural or behavioral test bench for the respective application. The test bench generated is based on this choice. This menu is applicable only to the IRST application.

4) Mode of operation :

This menu allows the user to choose between the file I/O option or the user interaction mode. Based on this menu the MODE input to the IRST test bench is set. This menu is applicable only to the IRST application.

As a part of this thesis, the menu structure was developed for generation of structural test benches and relates predominantly to the user interaction mode of operation. The user is prompted for some of the input values that are to be fed to the test bench model. In case of the IRST application, the values prompted for from the user are:

1) Initial X_COORD in the range 0 to 32. This value is assigned to the input port X_COORD in the test bench.

Chapter 7. Intelligent User Interface

- 2) Initial Y_COORD in the range 0 to 32. This value is assigned to the input port Y_COORD in the test bench.
- 3) Type of target motion. The options from which the user chooses from are: Horizontal, Vertical, Climb and Random. Based on the user choice a value is assigned to the input port MODE_MOTION of the test bench. If user's choice is horizontal, 1 is assigned to MODE_MOTION, if vertical, 2 is assigned to MODE_MOTION, if climb, 3 is assigned to MODE_MOTION, if random, 4 is assigned to MODE_MOTION.
- 4) If the type of target motion was chosen as Horizontal or Vertical the user is prompted for the velocity in the range 0 to 5 pixels per frame. If the type of target motion was chosen as Climb the user is first prompted for the angle of movement in the range 0 to 90 degrees, and then for the velocity in the range 0 to 5 pixels per frame.
- 5) The user is then prompted for the start time and end time of simulations, and the time between update of frames.
- 6) The user is then prompted for the choice between ON-LINE processing where the outputs are displayed at the end of every clock cycle using Synopsys simulator, and POST processing, where in the outputs are displayed at the end of simulation of all the frames using MATLAB.

Based on the input values keyed in by the user, the IUI creates a simulation control file which is used to control the simulation of the test bench model using the Synopsys simulator. The IUI forms the first few lines of the simulation control file consisting of assign statements to assign values keyed in by the user to the input ports of the test bench. The IUI forms the final simulation control file by concatenating the first few lines of the simulation control file formed with the templates of the simulation control files chosen based on the type of test bench (structural/behavioral) and the mode of processing (on-line

Chapter 7. Intelligent User Interface

and post processing) selected by the user. The templates of the simulation control files are precreated, and they control the sequencing of the simulations using the Synopsys simulator. Figure 7.2 depicts the template of the simulation control file created for a structural test bench operating in the post processing mode. As shown in the template the simulation is run in steps of time given by the simulation control variable PERIOD. The value of PERIOD is initialized based on the frame update rate keyed in by the user.

```
assign (7) TARG_RANGE
open /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/frame_no.out
open /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/comp.out

set FRCNT 1
set FRCNT1 1
set ZEROS 0

while ((INIT = '1') and (RUN = '0')) loop

    run $PERIOD

    set FRCNT `FRCNT+1`

    fprintf "%d " $ZEROS >
/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/comp.out
    fprintf "%d\n" $ZEROS >
/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/comp.out

    fprintf "%d\n" $FRCNT1 >
/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/frame_no.out
    set FRCNT1 `FRCNT1+1`

end loop

while (RUN = '1') loop

    run $PERIOD

    fprintf "%d " X_COORD >
/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/comp.out
    fprintf "%d\n" Y_COORD >
/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/comp.out
```

```
fprint "%d\n" $FRCNT1 >
/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/frame_no.out
set FRCNT1 `FRCNT1+1`

end loop

set FRCNT1 999

fprint "%d\n" $FRCNT1 >
/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/frame_no.out

!echo '999' >> /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/images.txt

quit
```

FIGURE 7.2 A Template of SCL File

Files "comp.out" and "frame_no.out" are created during simulation to interface the test bench with the model under test. In case of ON_LINE processing, the simulation control file template used in the simulation control file is used to display results on a Synopsys pad window. In case of Post processing, the sequence of image frames created by the test bench is fed by the IUI to a MATLAB program that is used to display the image frames in pixel bit map format using MATLAB. Other template files and the final simulation control file created by the IUI are attached in Appendix D.

Chapter 8. Results

8.1 Simulation Results

The structural test bench primitives developed in SGE and the structural test benches developed in SGE as schematics interconnecting the above primitives for the Infrared Search and Track (IRST) application and the Synthetic Aperture Radar (SAR) application were simulated using the Synopsys simulator [19]. The simulation control files created using the Intelligent User Interface based on user inputs controlled the simulation of the test bench models.

Simulation efficiency measurements were done to compare the CPU times taken for simulating both the signal based structural test bench model developed using SGE and the variable based procedural test bench model. Equivalent procedure based models were developed for the IRST primitives explained in Chapter 5. A variable based IRST test bench model was created using the equivalent procedure based components. Both the test

Chapter 8. Results

bench models were simulated and CPU times for simulation were recorded using a C program attached in Appendix E. Table 8.1 depicts the CPU times recorded for simulating both the models for the IRST test bench.

TABLE 8.1 **Table Depicting CPU Times Recorded for Simulating the Two IRST Testbench Models**

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

The above table shows that the procedure based test bench model simulates about 10 -20 % more efficiently than the entity based test bench model.

The output of the IRST testbench was a sequence of 32x32 two dimensional arrays, whose element values varied from 0 to 255. This sequence of arrays was written into a file for application to the MUT. The output of the SAR testbench was a sequence

Chapter 8. Results

of the sample values of the returned signal. The outputs were stored in a file, each line of which contained the real and imaginary parts of the returned signal.

8.2 Output Displays

Besides the output results generated by the test benches for the IRST and SAR applications, several displays were created for both the test benches in order to analyze the results.

8.2.1 IRST Test Bench

The sequence of arrays obtained after simulation were displayed using Synopsys or MATLAB depending on the mode of processing chosen by the user. The "pad window" feature of the Synopsys simulator [19] was used to display the output arrays when on-line processing was chosen. The display screen was refreshed every time the new output array was created. Figure 8.1 shows one such display created using the Synopsys simulator. This figure depicts the third output frame created by the IRST test bench, with the target at (13,14) shown by a "#" mark. The target position in the previous frame, i.e., (12,13) is shown by a "." mark.

For post-processing, MATLAB [20] was used to display the arrays. The arrays were read in from the file in which they were stored and displayed as a sequence of gray scale bit-maps. Finally, all the arrays were displayed together in a single window. Figure 8.2 shows the gray scale bit-map display of a single array. This figure depicts the three

clutter regions in different shades of gray and the target position is shown in white. Figure 8.3 shows the bit-map display of all the arrays in a single window.

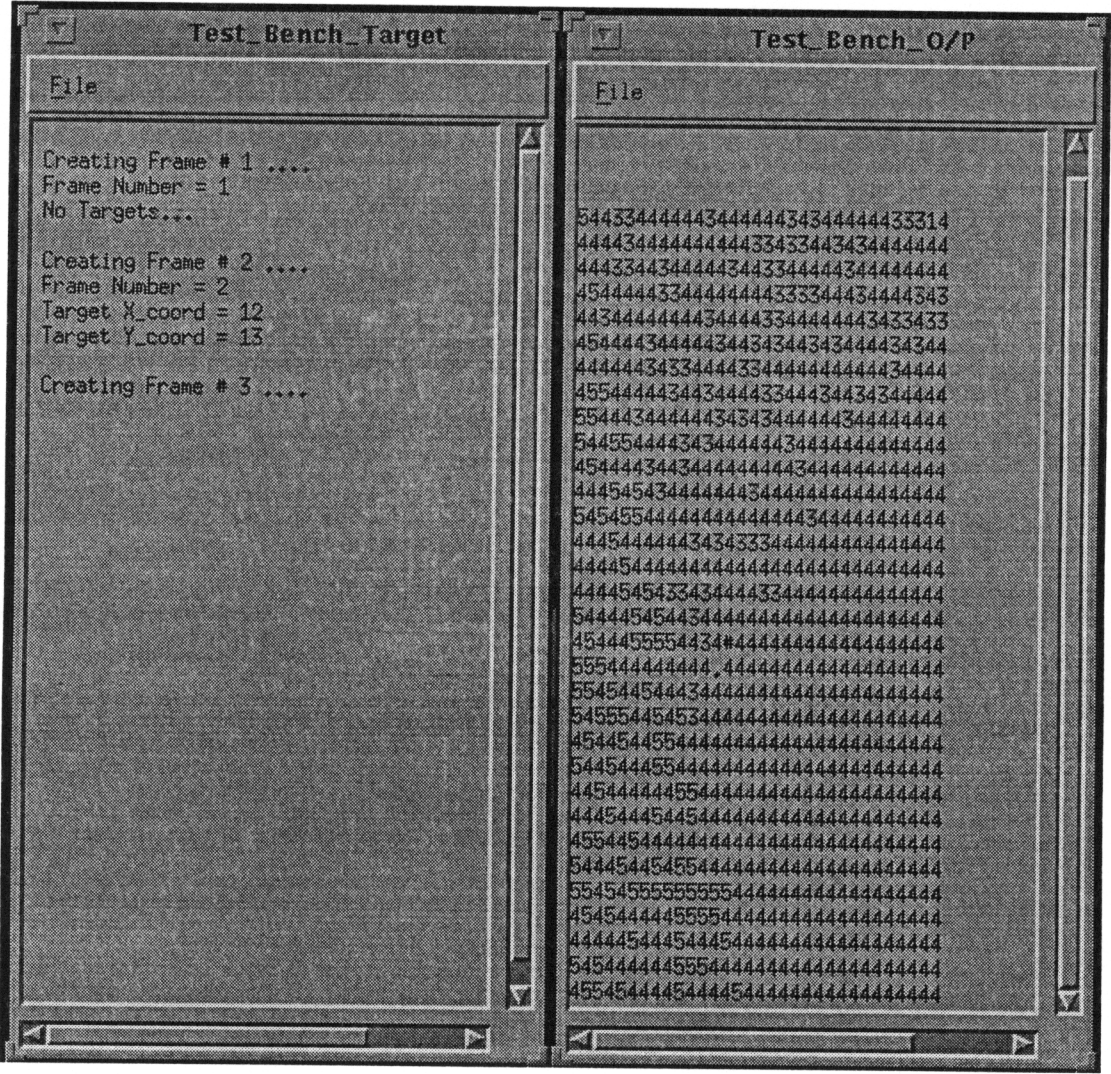


FIGURE 8.1 **Display of Output Frame using Synopsys**

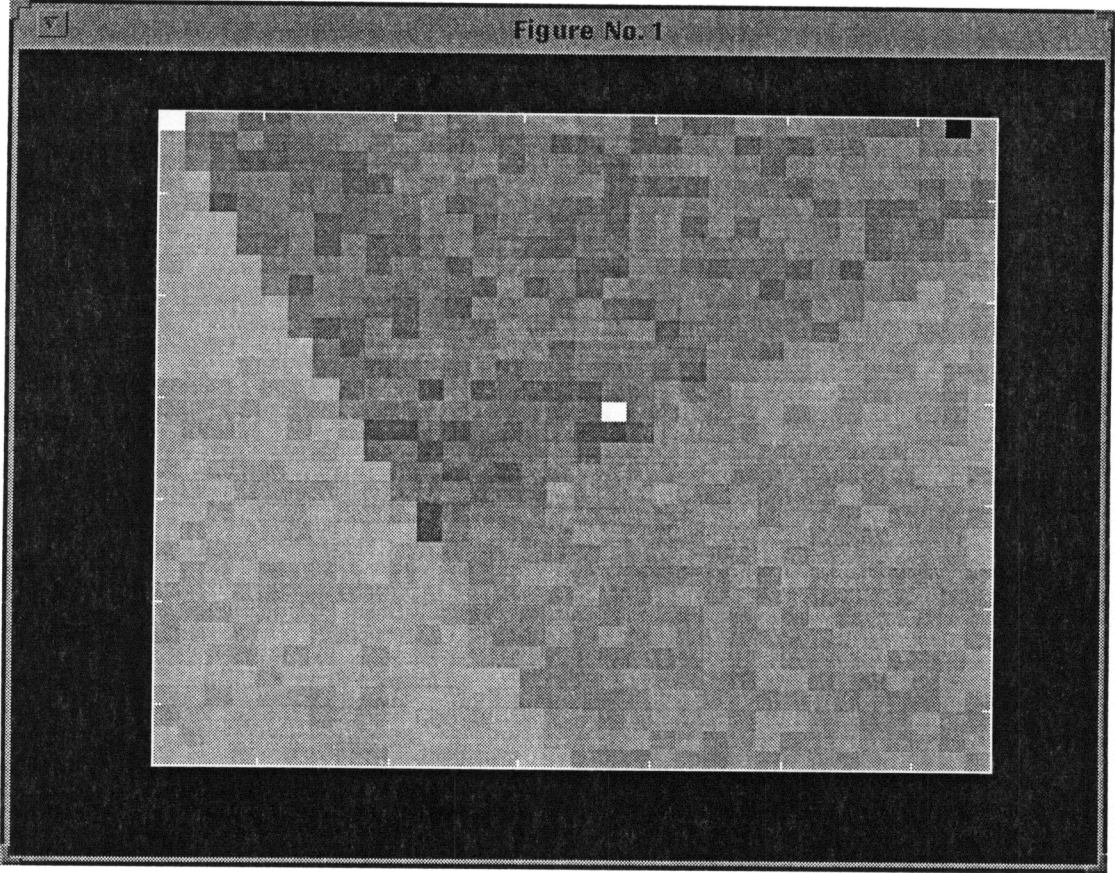


FIGURE 8.2 **Display of Output Frame using MATLAB**

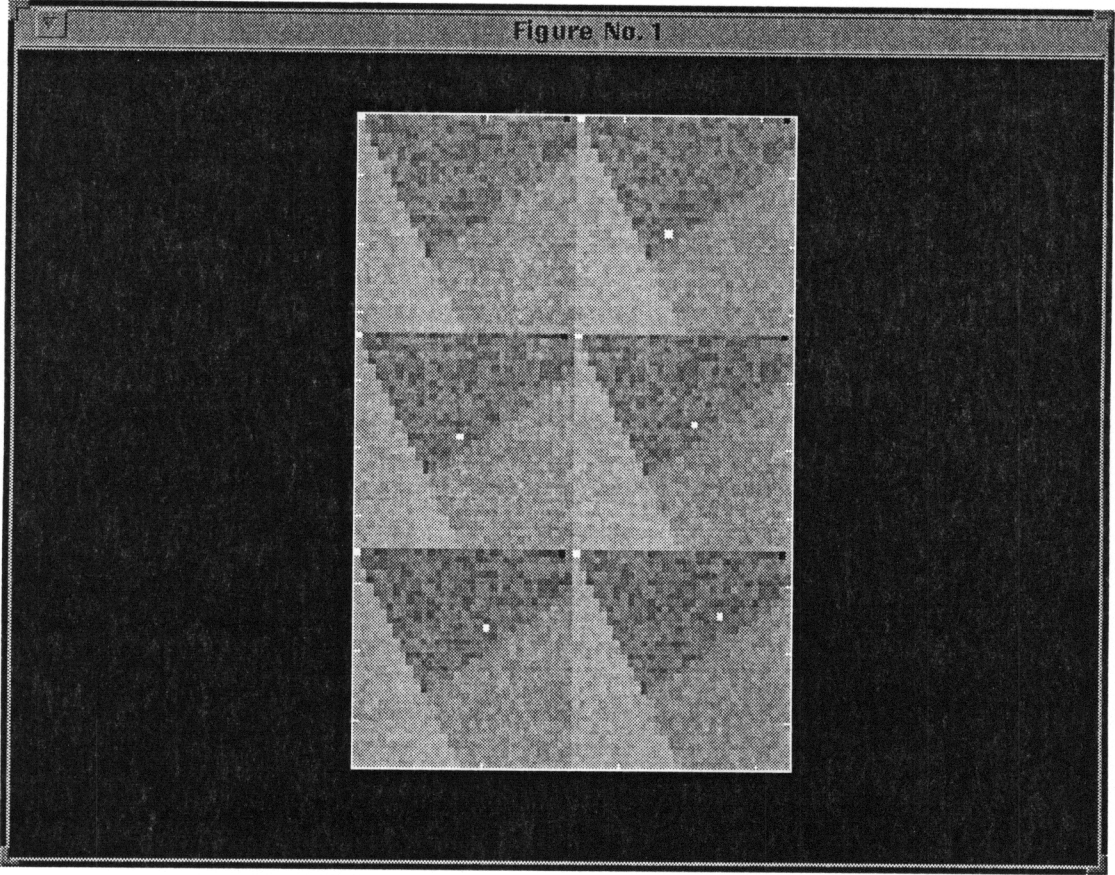


FIGURE 8.3. **Display of Sequence of Frames using MATLAB**

8.2.2 SAR Test Bench

MATLAB [20] was used to display the output signal. The files containing the real and imaginary parts of the sample values were read and then the signal was displayed as two plots, one showing the real part and the other showing the imaginary part of the returned signal. Figure 8.4 shows the plots generated by MATLAB. The initial delay in the sine wave shown below is due to the round trip delay between the transmitted and received signals. This delay gives a measure of the range. The frequency of the sine wave also gives a measure of the range.

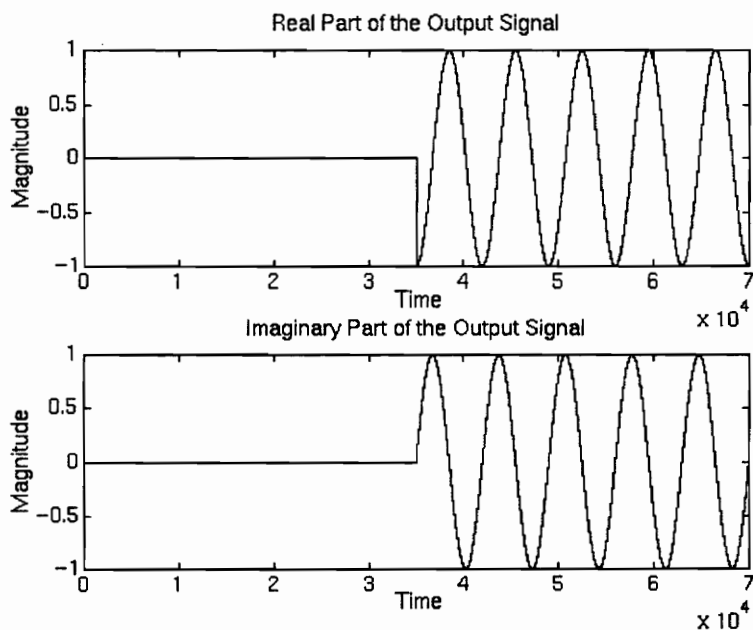


FIGURE 8.4. MATLAB Plot of the SAR Output Signal

Chapter 9. Conclusions and Future Work

This thesis has presented a methodology for structural test bench development for DSP based systems using the library of primitive components and Synopsys Graphical Environment (SGE), a conventional schematic capture tool. This approach has been implemented for two applications: 1) A 2D Image processing algorithm - InfraRed Search and Track (IRST) and 2) Synthetic Aperture Radar (SAR). A library of primitive components have been developed for the above two applications. Structural test benches have been developed using SGE and the library of primitives. The testbenches were simulated successfully and the results obtained were displayed and analyzed. A part of the Intelligent User Interface has been developed in this thesis to create a testbench generation system. A requirements capture interface has been developed for IRST using RDD-100, to generate generic values.

Some of the avenues in which future work is possible are as follows. The architectural part of the primitives could be automatically generated using CASE tools like I-LOGIX Express V-HDL. An application oriented directory structure could be developed

Chapter 9. Conclusions and Future Work

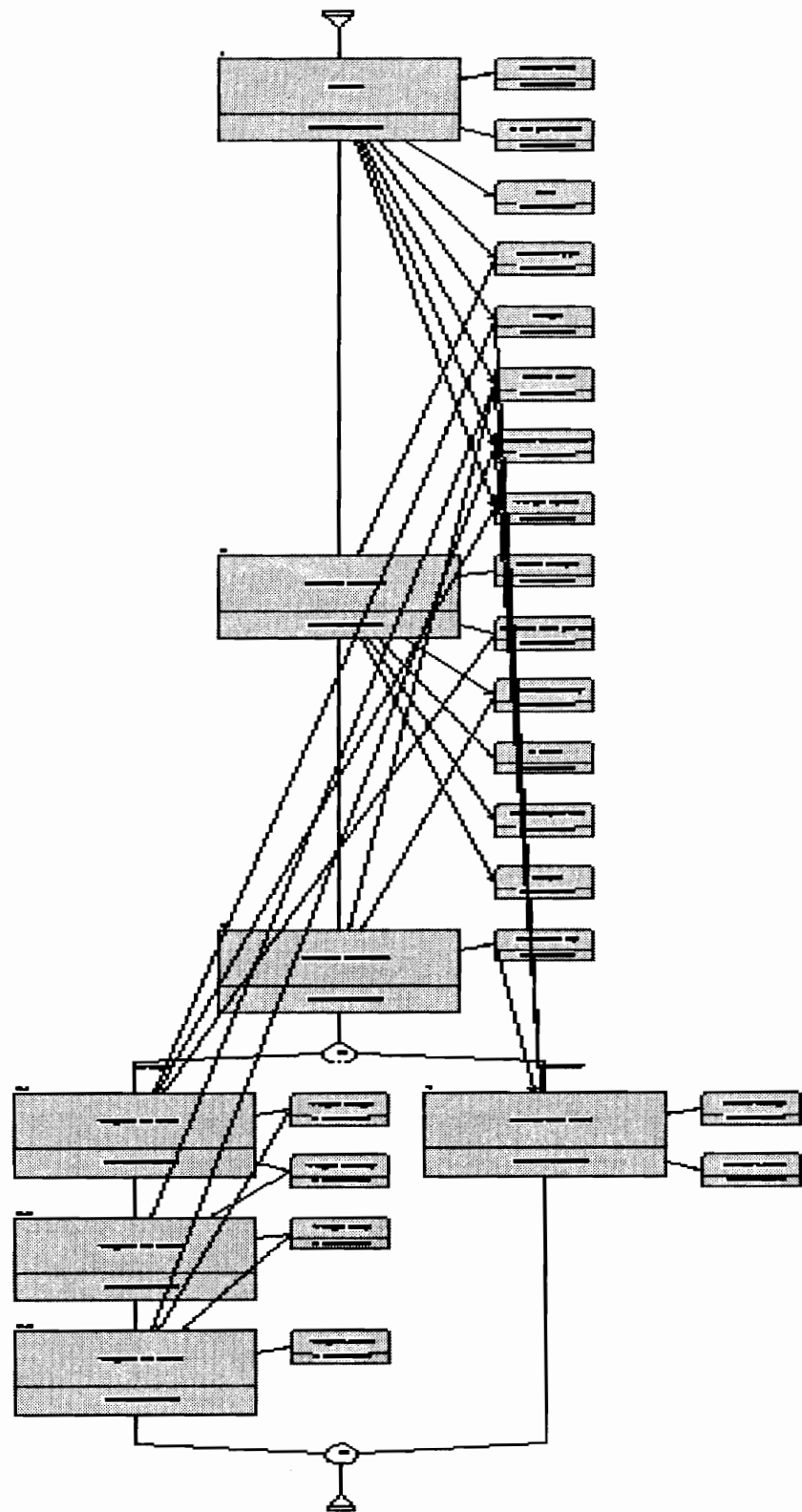
for the library of primitives, in order to enhance the automation of the test bench generation task. The event transcript file generated by RDD 100 during dynamic verification contains the values of the secondary requirements derived using the math model. This event transcript file can be parsed using a C program, and the values of the secondary requirements can be extracted and written to a file. As a part of the requirements interface, this file produced by the parser program containing the values of the secondary requirements can be used as input to the Intelligent User Interface (IUI). The Intelligent User Interface could be developed into an X-Window interface and could be made more user-friendly.

Bibliography

- [1] D. D. Gajski, N. D. Dutt, A. C-H Wu, S. Y-L Lin, "HIGH-LEVEL SYNTHESIS - Introduction to Chip and System Design," Kluwer Academic Publishers, 1992.
- [2] M. Donlin, "ASIC Complexity fuels drive to HDL design," *Computer Design*, May 1991.
- [3] D. D. Gajski, "Essential Issues and Possible Solutions in High-Level Synthesis," High Level VLSI Synthesis, Kluwer Academic Publishers, 1991.
- [4] J. R. Armstrong and F. G. Gray, "Structured Logic Design with VHDL," Prentice Hall, 1993.
- [5] IEEE Standard VHDL Language Reference Manual, 1988.
- [6] S. Rao, B-Y Pan and J. Armstrong, "Hierarchical Test Generation for VHDL Behavioral Models," *Proc. of EDAC*, pp. 175-182, 1993.
- [7] M.A. Richards, "The RASSP Program : Overview and Accomplishments", ARPA, Proc. of First Annual RASSP Conference, Arlington VA, August 1994.
- [8] J. R. Armstrong and G. A. Frank, "Testbench Generator Specification ", December 1994.
- [9] Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman, "Data Structures and Algorithms," Addison-Wesley Publishing Company, 1983.

- [10] C. H. Cho and J.R. Armstrong, "The B Algorithm: A Behavioral Test Generation Algorithm", to presented at the International Test Conference, Fall 1994
- [11] 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.
- [12] 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
- [13] George F. Lounge, Julian Olyansky, Jine S. Tseng, "Multisensor VHSIC Signal Processor Requirements Analysis", Mission Avionics Technology Department, Naval Air Development Center, September 1989.
- [14] Xu Zhen, "Modeling Signals and Sensors Using VHDL For SAR System", Master of Science thesis, Electrical Engineering Department, Virginia Tech, February 1995.
- [15] B. Zuerndorfer, G.A. Shaw, "SAR Processing for RASSP Application" , M.I.T. Lincoln Laboratory, Proc. of First Annual RASSP Conference, Arlington VA, August 1994.
- [16] G.A.Shaw, "Synthetic Aperture Radar Image Processor - RASSP Benchmark" , M.I.T. Lincoln Laboratory, January 1994.
- [17] Ascent Logic Corporation, RDD-100 User's Guide, Release 4.
- [18] Synopsys Graphical Environment User Guide, Version 3.0, December 1992.
- [19] Synopsys VHDL System Simulator - Command Reference Manual, Version 3.0, December 1992.
- [20] MATLAB Reference Guide, August 1992.

Appendix A: Behavior Diagram for Time Function Requirements



Appendix B: VHDL Code listing for IRST and SAR

Infrared Search and Track Primitives

Oscillator Component

-- VHDL Model Created from SGE Symbol osc22.sym -- Jul 27 13:24:57 1994

```
use work.all;
  use STD.TEXTIO.all;
  use work.USER_TYPES1.all;
  use work.MATH_REAL.all;

entity OSC22 is
  Port (  INIT : In  BIT;
         PER  : In  TIME;
         CLK  : Out BIT );
end OSC22;

architecture BEHAVIORAL of OSC22 is

begin

  process

  begin

    while INIT = '1' loop
      CLK <= '1';
      wait for PER/2;
      CLK <= '0';
```

Appendix B

```
wait for PER/2;
end loop;

end process;

end BEHAVIORAL;

configuration CFG_OSC22_BEHAVIORAL of OSC22 is
  for BEHAVIORAL

    end for;

end CFG_OSC22_BEHAVIORAL;
```

Environment Component

-- VHDL Model Created from SGE Symbol environ22.sym -- Nov 30 00:04:41 1994

```
use work.all;
use STD.TEXTIO.all;
use work.USER_TYPES1.all;
use work.MATH_REAL.all;

entity ENVIRON22 is
  Generic ( DEL: TIME:= 0 NS );
  Port (   CLK : In  BIT;
          CLUT : Out INT_2D(0 TO 31,0 TO 31) := (OTHERS => (OTHERS =>
                                                                0)) );
end ENVIRON22;

architecture BEHAVIORAL of ENVIRON22 is

begin

process

  variable VLINE: LINE;
  variable TEMP,CLUT_CNT: INTEGER:= 0;
  variable CLUT_TEMP: INT_2D(0 to 31,0 to 31):= (others => (others => 0));
  file INCLUTTER: TEXT is in
"/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/STRUCTURAL/inclutter.txt";

begin

  wait on CLK until CLK = '1';

  if not(ENDFILE(INCLUTTER)) then

    READLINE(INCLUTTER,VLINE);
```

Appendix B

```
    READ(VLINE, TEMP);

    CLUT_CNT := TEMP;

    if not(CLUT_CNT = 999) then

        for I in 0 to 31 loop
            READLINE(INCLUTTER,VLINE);
            for J in 0 to 31 loop
                READ(VLINE, TEMP);
                CLUT_TEMP(I,J) := TEMP;
            end loop;
        end loop;

    end if;

end if;

CLUT <= CLUT_TEMP after DEL ;

end process;

end BEHAVIORAL;

configuration CFG_ENVIRON22_BEHAVIORAL of ENVIRON22 is
    for BEHAVIORAL

        end for;
    end CFG_ENVIRON22_BEHAVIORAL;
```

Target Component

-- VHDL Model Created from SGE Symbol target22.sym -- Nov 30 00:06:00 1994

```
use work.all;
use STD.TEXTIO.all;
use work.USER_TYPES1.all;
use work.MATH_REAL.all;

entity TARGET22 is
    Generic(DEL:TIME:= 0 NS;
    REG1_RANGE : INTEGER := 5;
    REG2_RANGE : INTEGER := 10;
    REG3_RANGE: INTEGER := 15;
    FRAME_SIZE : INTEGER := 32 );
    Port (  ANGLE : In  INTEGER;
           CLK : In  BIT;
           INIT : In  BIT;
           MODE : In  BIT := '0';
           MODE_MOTION : In  INTEGER;
```

Appendix B

```

        RUN : In  BIT;
        TARG_RANGE : In  INTEGER;
        VELOCITY : In  INTEGER;
        X_COORD : InOut INTEGER;
        Y_COORD : InOut INTEGER;
        FRAME : Out  INT_2D(0 TO 31,0 TO 31) := (OTHERS => (OTHERS => 0));
end TARGET22;

architecture BEHAVIORAL of TARGET22 is

begin

process(INIT,CLK,RUN)

    variable FRAME_TEMP: INT_2D(0 to 31,0 to 31):= (others => (others => 0));
    variable VLINE1: LINE;
    variable ANGLE_RAD: REAL;
    variable TEMP_ANGLE: INTEGER;
    variable I: INTEGER:= 0;
    variable I1: INTEGER:= 1;
    variable J: INTEGER:= 0;
    variable NEW_X_COORD,NEW_Y_COORD:INTEGER:= 0;
    variable TEMP1: INTEGER:= 0;
    variable TEMP_X,TEMP_Y: INTEGER := 0;
    variable TAR_POS: INT_2D(0 to 20,0 to 1):= (others => (others => 0));
    variable FLAG1 : BIT := '0';
    variable START_FLAG : BIT := '0';
    variable RAND_NO1,RAND_NO2 : REAL;
    variable SEED1: INTEGER := 2000;
    variable SEED2: INTEGER := 5000;
    file POSITION : TEXT is in "/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/target.in";

begin

    if MODE = '0' then

        if INIT = '1' and RUN = '0' then
            NEW_X_COORD := X_COORD;
            NEW_Y_COORD := Y_COORD;
        end if;

        if RUN = '1' then
            if CLK'EVENT and CLK = '1' then

                FRAME_TEMP(TEMP_X,TEMP_Y) := 0;

                case MODE_MOTION is
                    when 1 => TEMP_ANGLE := 0;
                        FLAG1 := '0';
                    when 2 => TEMP_ANGLE := 90;
```

Appendix B

```
        FLAG1 := '0';
    when 3 => TEMP_ANGLE := ANGLE;
        FLAG1 := '0';
    when 4 =>
        UNIFORM(SEED1,SEED1,RAND_NO1);
        UNIFORM(SEED2,SEED2,RAND_NO2);
        FLAG1 := '1';
    when others => null;

end case;

if ((FLAG1 = '0') and (START_FLAG = '1')) then
    ANGLE_RAD:= MATH_DEG_TO_RAD*REAL(TEMP_ANGLE);
    NEW_X_COORD := NEW_X_COORD +
INTEGER(REAL(VELOCITY)*COS(ANGLE_RAD));
    NEW_Y_COORD := NEW_Y_COORD + INTEGER(REAL(VELOCITY)*SIN(ANGLE_RAD));
elseif ((FLAG1 = '1') and (START_FLAG = '1')) then
    NEW_X_COORD := INTEGER(31.0*RAND_NO1);
    NEW_Y_COORD := INTEGER(31.0*RAND_NO2);
end if;

if ((NEW_X_COORD<FRAME_SIZE) and (NEW_Y_COORD<FRAME_SIZE)) then

    X_COORD <= NEW_X_COORD;
    Y_COORD <= NEW_Y_COORD;

    TEMP_X := 31-NEW_Y_COORD;
    TEMP_Y := NEW_X_COORD;

    if (15*NEW_Y_COORD <= (-31*NEW_X_COORD+465)) then
        if REG1_RANGE > TARG_RANGE then
            FRAME_TEMP(TEMP_X,TEMP_Y) := 255 ;
        end if;
    elseif ((7*NEW_Y_COORD <= (5*NEW_X_COORD+20)) and (15*NEW_Y_COORD > (-
31*NEW_X_COORD+465))) then
        if REG2_RANGE > TARG_RANGE then
            FRAME_TEMP(TEMP_X,TEMP_Y) := 255 ;
        end if;
    else
        if REG3_RANGE > TARG_RANGE then
            FRAME_TEMP(TEMP_X,TEMP_Y) := 255 ;
        end if;
    end if;

FRAME <= FRAME_TEMP after DEL;

START_FLAG := '1';
```

Appendix B

```
    end if;
  end if;
end if;

elsif MODE = '1' then

  if INIT = '1' and RUN = '0' then

    while not(ENDFILE(POSITION)) loop

      READLINE(POSITION,VLINE1);

      for J in 0 to 1 loop
        READ(VLINE1, TEMP1);
        TAR_POS(I,J):= TEMP1;
      end loop;

      I:= I+1;

    end loop;

    NEW_X_COORD := TAR_POS(0,0);
    NEW_Y_COORD := TAR_POS(0,1);

  end if;

  if RUN = '1' then
    if CLK'EVENT and CLK = '1' then

      FRAME_TEMP(TEMP_X,TEMP_Y) := 0;

      if ((NEW_X_COORD<FRAME_SIZE) and (NEW_Y_COORD<FRAME_SIZE)) then

        X_COORD <= NEW_X_COORD;
        Y_COORD <= NEW_Y_COORD;

        TEMP_X:= 31-NEW_Y_COORD;
        TEMP_Y := NEW_X_COORD;

        if (15*NEW_Y_COORD <= (-31*NEW_X_COORD+465)) then
          if REG1_RANGE > TARG_RANGE then
            FRAME_TEMP(TEMP_X,TEMP_Y) := 255 ;
          end if;
        elsif ((7*NEW_Y_COORD <= (5*NEW_X_COORD+20)) and (15*NEW_Y_COORD > (-
31*NEW_X_COORD+465))) then
          if REG2_RANGE > TARG_RANGE then
            FRAME_TEMP(TEMP_X,TEMP_Y) := 255 ;
          end if;
        end if;
      end if;
    end if;
  end if;
end if;
```

Appendix B

```

        end if,
    else
        if REG3_RANGE > TARG_RANGE then
            FRAME_TEMP(TEMP_X,TEMP_Y) := 255 ;
        end if,
    end if,

    FRAME <= FRAME_TEMP after DEL;

end if,

if not((TAR_POS(I1,0)=0) and (TAR_POS(I1,1)=0)) then
    NEW_X_COORD := TAR_POS(I1,0);
    NEW_Y_COORD := TAR_POS(I1,1);
    I1 := I1+1;
end if;

end if;
end if;
end if;

end process;

end BEHAVIORAL;

configuration CFG_TARGET22_BEHAVIORAL of TARGET22 is
    for BEHAVIORAL
        end for;
end CFG_TARGET22_BEHAVIORAL;

Sensor Component

-- VHDL Model Created from SGE Symbol sensor22.sym -- Nov 30 00:05:27 1994

use work.all;
use STD.TEXTIO.all;
use work.USER_TYPES1.all;
use work.MATH_REAL.all;

entity SENSOR22 is
    Generic ( DEL:TIME:= 0 NS );
    Port (CLK : In  BIT;
          CLUT : In  INT_2D(0 TO 31,0 TO 31) := (OTHERS => (OTHERS => 0));
          FRAME : In  INT_2D(0 TO 31,0 TO 31) := (OTHERS => (OTHERS => 0));
          INIT : In  BIT;
          IMAGE : Out INT_2D(0 TO 31,0 TO 31) := (OTHERS => (OTHERS => 0)) );
end SENSOR22;
```

Appendix B

architecture BEHAVIORAL of SENSOR22 is

begin

process

```
variable I: INTEGER:= 0;
variable J: INTEGER:= 0;
variable COUNT: INTEGER:= 1;
variable COUNT1: INTEGER:= 999;
variable IMAGE_TEMP: INT_2D(0 to 31,0 to 31):= (others => (others => 0));
variable VLINE2: LINE;
variable SPACE: CHARACTER:= ' ';
file IMAGES : TEXT is out "/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/images.txt";
```

begin

wait on INIT, CLK until CLK = '1';

if INIT = '1' then

wait for DEL;

for I in 0 to 31 loop

for J in 0 to 31 loop

if ((CLUT(I,J)=255) or (FRAME(I,J)=255)) then

IMAGE_TEMP(I,J) := 255;

else

IMAGE_TEMP(I,J) := CLUT(I,J) + FRAME(I,J);

end if;

IMAGE(I,J) <= IMAGE_TEMP(I,J);

end loop;

end loop;

WRITE(VLINE2,COUNT);

WRITELINE(IMAGES,VLINE2);

for I in 0 to 31 loop

for J in 0 to 31 loop

WRITE(VLINE2,IMAGE_TEMP(I,J));

WRITE(VLINE2,SPACE);

end loop;

WRITELINE(IMAGES,VLINE2);

end loop;

COUNT:= COUNT + 1;

Appendix B

```
    end if;

    end process;

end BEHAVIORAL;

configuration CFG_SENSOR22_BEHAVIORAL of SENSOR22 is
    for BEHAVIORAL

        end for;

end CFG_SENSOR22_BEHAVIORAL;
```

Infrared Search and Track Testbench

-- VHDL Model Created from SGE Schematic tb22.sch -- Nov 30 00:10:17 1994

```
use work.all;
use STD.TEXTIO.all;
use work.USER_TYPES1.all;
use work.MATH_REAL.all;
```

entity TB22 is

```
    Port ( ANGLE : In  INTEGER;
           INIT : In  BIT;
           MODE : In  BIT;
           MODE_MOTION : In  INTEGER;
           PER : In  TIME;
           RUN : In  BIT;
           TARG_RANGE : In  INTEGER;
           VELOCITY : In  INTEGER;
           X_COORD : InOut INTEGER;
           Y_COORD : InOut INTEGER;
           IMAGE : Out  INT_2D(0 TO 31,0 TO 31) );
end TB22;
```

architecture SCHEMATIC of TB22 is

```
    signal FRAME : INT_2D(0 TO 31,0 TO 31);
    signal CLUT : INT_2D(0 TO 31,0 TO 31);
    signal CLK : BIT;
```

component SENSOR22

Generic (DEL_TIME:= 0 NS);

```
    Port ( CLK : In  BIT;
           CLUT : In  INT_2D(0 TO 31,0 TO 31) := (OTHERS => (OTHERS => 0));
           FRAME : In  INT_2D(0 TO 31,0 TO 31) := (OTHERS => (OTHERS => 0));
           INIT : In  BIT;
```

Appendix B

```
    IMAGE : Out  INT_2D(0 TO 31,0 TO 31) := (OTHERS => (OTHERS => 0)) );  
end component;
```

```
component TARGET22
```

```
Generic ( DEL:TIME:= 0 NS;
```

```
    REG1_RANGE : INTEGER := 5;
```

```
    REG2_RANGE : INTEGER := 10;
```

```
    REG3_RANGE: INTEGER := 15;
```

```
    FRAME_SIZE : INTEGER := 32 );
```

```
Port (  ANGLE : In  INTEGER;
```

```
    CLK : In  BIT;
```

```
    INIT : In  BIT;
```

```
    MODE : In  BIT := '0';
```

```
    MODE_MOTION : In  INTEGER;
```

```
    RUN : In  BIT;
```

```
    TARG_RANGE : In  INTEGER;
```

```
    VELOCITY : In  INTEGER;
```

```
    X_COORD : InOut INTEGER;
```

```
    Y_COORD : InOut INTEGER;
```

```
    FRAME : Out  INT_2D(0 TO 31,0 TO 31) := (OTHERS => (OTHERS => 0)) );
```

```
end component;
```

```
component ENVIRON22
```

```
Generic ( DEL: TIME:= 0 NS );
```

```
Port (  CLK : In  BIT;
```

```
    CLUT : Out  INT_2D(0 TO 31,0 TO 31) := (OTHERS => (OTHERS => 0)) );
```

```
end component;
```

```
component OSC22
```

```
Port (  INIT : In  BIT;
```

```
    PER : In  TIME;
```

```
    CLK : Out  BIT );
```

```
end component;
```

```
begin
```

```
I_1 : SENSOR22
```

```
Generic Map ( DEL => 1 NS )
```

```
Port Map ( CLK=>CLK, CLUT=>CLUT, FRAME=>FRAME, INIT=>INIT,
```

```
    IMAGE=>IMAGE );
```

```
I_2 : TARGET22
```

```
Generic Map ( DEL => 1 NS, REG1_RANGE => 5, REG2_RANGE => 10,
```

```
    REG3_RANGE=> 15, FRAME_SIZE => 32 )
```

```
Port Map ( ANGLE=>ANGLE, CLK=>CLK, INIT=>INIT, MODE=>MODE,
```

```
    MODE_MOTION=>MODE_MOTION, RUN=>RUN,
```

```
    TARG_RANGE=>TARG_RANGE, VELOCITY=>VELOCITY,
```

```
    X_COORD=>X_COORD, Y_COORD=>Y_COORD, FRAME=>FRAME );
```

```
I_3 : ENVIRON22
```

```
Generic Map ( DEL => 1 NS )
```

```
Port Map ( CLK=>CLK, CLUT=>CLUT );
```

```
I_4 : OSC22
```

Appendix B

```
Port Map ( INIT=>INIT, PER=>PER, CLK=>CLK );

end SCHEMATIC;

configuration CFG_TB22_SCHEMATIC of TB22 is

  for SCHEMATIC
    for I_1: SENSOR22
      use configuration WORK.CFG_SENSOR22_BEHAVIORAL;
    end for;
    for I_2: TARGET22
      use configuration WORK.CFG_TARGET22_BEHAVIORAL;
    end for;
    for I_3: ENVIRON22
      use configuration WORK.CFG_ENVIRON22_BEHAVIORAL;
    end for;
    for I_4: OSC22
      use configuration WORK.CFG_OSC22_BEHAVIORAL;
    end for;
  end for;

end CFG_TB22_SCHEMATIC;
```

Synthetic Aperture Radar Primitives

Chirp Component

-- VHDL Model Created from SGE Symbol chirp.sym -- Aug 15 15:06:17 1994

```
use work.all;
use work.MATH_REAL.all;
```

```
entity CHIRP is
  Port (   BW : In  REAL;
          N : In  REAL;
          PRF : In  REAL;
          PW : In  REAL;
          SAMP_FREQ : In  REAL;
          I : Out REAL;
          R : Out REAL );
end CHIRP;
```

architecture BEHAVIORAL of CHIRP is

begin

```
  process
    variable k:  REAL := 0.0;
```

Appendix B

```
variable period:      REAL := 0.0;
variable curtime:     REAL := 0.0;
variable curval:      REAL := 0.0;
begin
    wait on N;
    k := bw / pw;
    period := 1.0 / prf;
    curtime := N / samp_freq;
    curval := curtime - floor ( curtime / period ) * period;
    if curval < pw then
        R <= cos ( 180.0 * k * N * N / samp_freq / samp_freq );
        I <= sin ( 180.0 * k * N * N / samp_freq / samp_freq );
    else
        R <= 1.0;
        I <= 0.0;
    end if;
end process;

end BEHAVIORAL;

configuration CFG_CHIRP_BEHAVIORAL of CHIRP is
    for BEHAVIORAL

        end for;

end CFG_CHIRP_BEHAVIORAL;
```

Comptfp Component

-- VHDL Model Created from SGE Symbol comptfp.sym -- Aug 15 15:33:38 1994

```
use work.all;
use work.MATH_REAL.all;

entity COMPTFP is
    Port (  FREQ : In  REAL;
            N : In  REAL;
            SAMP_FREQ : In  REAL;
            I : Out  REAL;
            R : Out  REAL );
end COMPTFP;

architecture BEHAVIORAL of COMPTFP is

begin

    process
    begin
        wait on N;
```

Appendix B

```
        R <= cos ( 2.0 * 180.0 * freq * N / samp_freq );
        I <= sin ( 2.0 * 180.0 * freq * N / samp_freq );
    end process;

end BEHAVIORAL;

configuration CFG_COMPTFP_BEHAVIORAL of COMPTFP is
    for BEHAVIORAL

        end for;

end CFG_COMPTFP_BEHAVIORAL;
```

Compmul Component

-- VHDL Model Created from SGE Symbol compmul.sym -- Aug 15 14:50:25 1994

```
use work.all;
use work.MATH_REAL.all;

entity COMPMUL is
    Port ( IIN1 : In  REAL;
           IIN2 : In  REAL;
           RIN1 : In  REAL;
           RIN2 : In  REAL;
           IOUT : Out REAL;
           ROUT : Out REAL );
end COMPMUL;

architecture BEHAVIORAL of COMPMUL is

begin

    process
        variable rv1: REAL := 0.0;
        variable rv2: REAL := 0.0;
        variable iv1: REAL := 0.0;
        variable iv2: REAL := 0.0;
    begin
        wait on Rin1, Iin1, Rin2, Iin2 ;
        if Rin1 <= -1.0e-35 then
            rv1 := -1.0e-35;
        else
            rv1 := Rin1;
        end if;
        if Rin2 <= -1.0e-35 then
            rv2 := -1.0e-35;
        else
            rv2 := Rin2;
        end if;
```

Appendix B

```
        if Iin1 <= -1.0e-35 then
            iv1 := -1.0e-35;
        else
            iv1 := Iin1;
        end if;
        if Iin2 <= -1.0e-35 then
            iv2 := -1.0e-35;
        else
            iv2 := Iin2;
        end if;
        Rout <= rv1 * rv2 - iv1 * iv2;
        Iout <= rv1 * iv2 + rv2 * iv1;
    end process;

end BEHAVIORAL;

configuration CFG_COMPMUL_BEHAVIORAL of COMPMUL is
    for BEHAVIORAL

        end for;

end CFG_COMPMUL_BEHAVIORAL;
```

Genchirp Component

-- VHDL Model Created from SGE Schematic genchirp.sch -- Aug 15 15:48:24 1994

```
use work.all;
use work.MATH_REAL.all;

entity GENCHIRP is
    Port (    BW : In  REAL;
            FREQ : In  REAL;
            N : In  REAL;
            PRF : In  REAL;
            PW : In  REAL;
            SAMP_FREQ : In  REAL;
            I : Out REAL;
            R : Out REAL );
end GENCHIRP;

architecture SCHEMATIC of GENCHIRP is

    signal    I2 : REAL;
    signal    R2 : REAL;
    signal    I1 : REAL;
    signal    R1 : REAL;

    component COMPTFP
        Port (    FREQ : In  REAL;
```

Appendix B

```

        N : In  REAL;
        SAMP_FREQ : In  REAL;
        I : Out  REAL;
        R : Out  REAL );
end component;

component COMPMUL
  Port (  IIN1 : In  REAL;
          IIN2 : In  REAL;
          RIN1 : In  REAL;
          RIN2 : In  REAL;
          IOUT : Out REAL;
          ROUT : Out REAL );
end component;

component CHIRP
  Port (  BW : In  REAL;
          N : In  REAL;
          PRF : In  REAL;
          PW : In  REAL;
          SAMP_FREQ : In  REAL;
          I : Out  REAL;
          R : Out  REAL );
end component;

begin

I_1 : COMPTFP
  Port Map ( FREQ=>FREQ, N=>N, SAMP_FREQ=>SAMP_FREQ, I=>I2, R=>R2 );
I_2 : COMPMUL
  Port Map ( IIN1=>I1, IIN2=>I2, RIN1=>R1, RIN2=>R2, IOUT=>I,
            ROUT=>R );
I_3 : CHIRP
  Port Map ( BW=>BW, N=>N, PRF=>PRF, PW=>PW, SAMP_FREQ=>SAMP_FREQ,
            I=>I1, R=>R1 );

end SCHEMATIC;

configuration CFG_GENCHIRP_SCHEMATIC of GENCHIRP is

  for SCHEMATIC
    for I_1: COMPTFP
      use configuration WORK.CFG_COMPTFP_BEHAVIORAL;
    end for;
    for I_2: COMPMUL
      use configuration WORK.CFG_COMPMUL_BEHAVIORAL;
    end for;
    for I_3: CHIRP
      use configuration WORK.CFG_CHIRP_BEHAVIORAL;
    end for;
  end for;

end for;
```

Appendix B

```
end CFG_GENCHIRP_SCHEMATIC;
```

Delay Component

```
-- VHDL Model Created from SGE Symbol delay.sym -- Aug 16 12:11:01 1994
```

```
use work.all;  
use work.MATH_REAL.all;
```

```
entity DELAY is  
  Generic ( DEL:INTEGER:= 35000 );  
  Port (    X : In  REAL;  
          Y : Out REAL );  
end DELAY;
```

```
architecture BEHAVIORAL of DELAY is
```

```
type REAL_VECTOR is array (NATURAL range <>) of REAL;
```

```
begin
```

```
  process  
    variable V: REAL_VECTOR ( 0 to DEL-1 ) := ( others => 0.0);  
  begin  
    wait on X;  
    Y <= V (DEL-1);  
    for i in DEL-1 downto 1 loop  
      V(i) := V(i-1);  
    end loop;  
    V(0) := X;  
  end process;
```

```
end BEHAVIORAL;
```

```
configuration CFG_DELAY_BEHAVIORAL of DELAY is  
  for BEHAVIORAL
```

```
    end for;
```

```
end CFG_DELAY_BEHAVIORAL;
```

Neg Component

```
-- VHDL Model Created from SGE Symbol neg.sym -- Aug 16 12:14:45 1994
```

```
use work.all;  
use work.MATH_REAL.all;
```

Appendix B

entity NEG is

```
Port (  IX : In  REAL;
        RX : In  REAL;
        IY : Out REAL;
        RY : Out REAL );
```

end NEG;

architecture BEHAVIORAL of NEG is

begin

```
process
begin
    wait on IX ;
    RY <= RX;
    IY <= -IX;
end process;
```

end BEHAVIORAL;

configuration CFG_NEG_BEHAVIORAL of NEG is
for BEHAVIORAL

end for;

end CFG_NEG_BEHAVIORAL;

Decimate Component

-- VHDL Model Created from SGE Symbol decimate.sym -- Aug 16 15:40:06 1994

use work.all;

use work.MATH_REAL.all;

entity DECIMATE is

```
Generic ( DEC:INTEGER );
Port (  IX : In  REAL;
        N : In  REAL;
        RX : In  REAL;
        IY : Out REAL;
        RY : Out REAL );
```

end DECIMATE;

architecture BEHAVIORAL of DECIMATE is

begin

```
process
    variable cnt: integer :=0;
```

Appendix B

```
begin
  wait on N;
  if N = 1.0 then
    RY <= RX;
    IY <= IX;
  end if;
  cnt := integer(N) rem DEC;
  if cnt = 0 then
    RY <= RX;
    IY <= IX;
  end if;
end process;

end BEHAVIORAL;

configuration CFG_DECIMATE_BEHAVIORAL of DECIMATE is
  for BEHAVIORAL

    end for;

end CFG_DECIMATE_BEHAVIORAL;
```

Change Component

-- VHDL Model Created from SGE Symbol change.sym -- Aug 24 16:27:18 1994

```
use work.all;
use work.MATH_REAL.all;

entity CHANGE is
  Generic ( TBIT: INTEGER;
            IBIT: INTEGER );
  Port (   RIN : In  REAL;
          BOUT : Out BIT_VECTOR(TBIT-1 DOWNT0 0) );
end CHANGE;
```

architecture BEHAVIORAL of CHANGE is

```
function INT_TO_BIN
  (INPUT: INTEGER; N: INTEGER)
  return BIT_VECTOR is
  variable FOUT: BIT_VECTOR( N-1 downto 0);
  variable TEMP_A: INTEGER := 0;
  variable TEMP_B: INTEGER := 0;
begin
  TEMP_A := INPUT;
  for i in N-1 downto 0 loop
    TEMP_B := TEMP_A/(2**I);
    TEMP_A := TEMP_A rem (2 **I);
```

Appendix B

```
        if ( TEMP_B =1) then
            FOUT(i) := '1';
        else
            FOUT(i) := '0';
        end if;
    end loop;
    return FOUT;
end INT_TO_BIN;
```

```
function INC (X: BIT_VECTOR)
    return BIT_VECTOR is
        variable XV: BIT_VECTOR (X'LENGTH-1 downto 0);
    begin
        XV := X;
        for i in 0 to XV'HIGH loop
            if XV(i) = '0' then
                XV(i) := '1';
                exit;
            else
                XV(i) := '0';
            end if;
        end loop;
        return XV;
    end INC;
```

```
function INVB (X: BIT_VECTOR)
    return BIT_VECTOR is
        variable XV: BIT_VECTOR (X'LENGTH-1 downto 0);
    begin
        XV := X;
        for i in 0 to XV'HIGH loop
            if XV(i) = '0' then
                XV(i) := '1';
            else
                XV(i) := '0';
            end if;
        end loop;
        return XV;
    end INVB;
```

```
begin
```

```
    L1: process
        variable INT: INTEGER := 0;
        variable DEC: INTEGER := 0;
        variable fr: REAL := 0.0;
        variable abrin: REAL := 0.0;
        variable TEMP: BIT_VECTOR(TBIT-2 downto 0);
        variable TEMPB: BIT_VECTOR(TBIT-2 downto 0);
    begin
```

Appendix B

```
wait on RIN;
if RIN < 0.0 then
    BOUT(TBIT-1) <= '1';
else
    BOUT(TBIT-1) <= '0';
end if;
abrin := abs(RIN);
fr := floor(abrin);
INT := integer(fr);
if IBIT /= 0 then
    TEMP(TBIT-2 downto TBIT-IBIT-1) := INT_TO_BIN(INT, IBIT);
end if;
DEC := integer((abrin-fr) / (1.0/256.0));
TEMP(TBIT-IBIT-2 downto 0) := INT_TO_BIN(DEC, TBIT-IBIT-1);
if RIN < 0.0 then
    TEMPB := INVB(TEMP);
    BOUT(TBIT-2 downto 0) <= INC(TEMPB) ;
else
    BOUT(TBIT-2 downto 0) <= TEMP;
end if;
end process L1;
```

end BEHAVIORAL;

configuration CFG_CHANGE_BEHAVIORAL of CHANGE is
for BEHAVIORAL

end for;

end CFG_CHANGE_BEHAVIORAL;

Pass Component

-- VHDL Model Created from SGE Symbol pass.sym -- Aug 16 12:16:40 1994

```
use work.all;
use work.MATH_REAL.all;
```

entity PASS is

```
Port (  IX : In  REAL;
        RX : In  REAL;
        IY : Out REAL;
        RY : Out REAL );
```

end PASS;

architecture BEHAVIORAL of PASS is

begin

process

Appendix B

```
begin
    wait on RX, IX ;
    RY <= RX;
    IY <= IX;
    end process;

end BEHAVIORAL;

configuration CFG_PASS_BEHAVIORAL of PASS is
    for BEHAVIORAL

        end for;

end CFG_PASS_BEHAVIORAL;
```

Synthetic Aperture Radar Testbench

-- VHDL Model Created from SGE Schematic sar_tb.sch -- Aug 16 16:58:42 1994

```
use work.all;
use work.MATH_REAL.all;

entity SAR_TB is
    Generic ( TBIT: IN INTEGER:= 12;
              IBIT: IN INTEGER:= 2;
              DEC:IN INTEGER:=1 );
    Port (    BW : In  REAL;
              FREQ : In  REAL;
              N : In  REAL;
              PRF : In  REAL;
              PW : In  REAL;
              SAMP_FREQ : In  REAL;
              IRH : InOut REAL;
              IRV : InOut REAL;
              RRH : InOut REAL;
              RRV : InOut REAL;
              IBH : Out  BIT_VECTOR(TBIT-1 DOWNT0 0);
              IBV : Out  BIT_VECTOR(TBIT-1 DOWNT0 0);
              RBH : Out  BIT_VECTOR(TBIT-1 DOWNT0 0);
              RBV : Out  BIT_VECTOR(TBIT-1 DOWNT0 0) );
end SAR_TB;
```

architecture SCHEMATIC of SAR_TB is

```
signal  IRV1 : REAL;
signal  RRV1 : REAL;
signal  DR1 : REAL;
signal  R1 : REAL;
```

Appendix B

```
signal    I1 : REAL;
signal    DI1 : REAL;
signal    NR2 : REAL;
signal    R2 : REAL;
signal    R4 : REAL;
signal    NR4 : REAL;
signal    I2 : REAL;
signal    NI2 : REAL;
signal    I4 : REAL;
signal    NI4 : REAL;
signal    RRH1 : REAL;
signal    IRH1 : REAL;
signal    R5 : REAL;
signal    R3 : REAL;
signal    NR3 : REAL;
signal    PR3 : REAL;
signal    I5 : REAL;
signal    I3 : REAL;
signal    NI3 : REAL;
signal    PI3 : REAL;
signal    R7 : REAL;
signal    R6 : REAL;
signal    NR6 : REAL;
signal    PR6 : REAL;
signal    I7 : REAL;
signal    I6 : REAL;
signal    NI6 : REAL;
signal    PI6 : REAL;
```

component CHANGE

```
Generic ( TBIT: INTEGER;
          IBIT:INTEGER );
  Port (  RIN : In  REAL;
          BOUT : Out BIT_VECTOR(TBIT-1 DOWNT0 0) );
end component;
```

component PASS

```
  Port (  IX : In  REAL;
          RX : In  REAL;
          IY : Out REAL;
          RY : Out REAL );
end component;
```

component NEG

```
  Port (  IX : In  REAL;
          RX : In  REAL;
          IY : Out REAL;
          RY : Out REAL );
end component;
```

component GENCHIRP

Appendix B

```
Port ( BW : In REAL;
      FREQ : In REAL;
      N : In REAL;
      PRF : In REAL;
      PW : In REAL;
      SAMP_FREQ : In REAL;
      I : Out REAL;
      R : Out REAL );
end component;

component DELAY
Generic ( DEL:INTEGER:= 35000 );
Port ( X : In REAL;
      Y : Out REAL );
end component;

component DECIMATE
Generic ( DEC:INTEGER );
Port ( IX : In REAL;
      N : In REAL;
      RX : In REAL;
      IY : Out REAL;
      RY : Out REAL );
end component;

component COMPTFP
Port ( FREQ : In REAL;
      N : In REAL;
      SAMP_FREQ : In REAL;
      I : Out REAL;
      R : Out REAL );
end component;

component COMPMUL
Port ( IIN1 : In REAL;
      IIN2 : In REAL;
      RIN1 : In REAL;
      RIN2 : In REAL;
      IOUT : Out REAL;
      ROUT : Out REAL );
end component;

begin

I_16 : CHANGE
Generic Map ( TBIT => TBIT, IBIT => IBIT )
Port Map ( RIN=>IRH, BOUT=>IBH );
I_25 : CHANGE
Generic Map ( TBIT => TBIT, IBIT => IBIT )
Port Map ( RIN=>RRV, BOUT=>RBV );
I_24 : CHANGE
```

Appendix B

```
Generic Map ( TBIT => TBIT, IBIT => IBIT )
  Port Map ( RIN=>IRV, BOUT=>IBV );
I_17 : CHANGE
Generic Map ( TBIT => TBIT, IBIT => IBIT )
  Port Map ( RIN=>RRH, BOUT=>RBH );
I_19 : PASS
  Port Map ( IX=>NI6, RX=>NR6, IY=>PI6, RY=>PR6 );
I_9 : PASS
  Port Map ( IX=>NI3, RX=>NR3, IY=>PI3, RY=>PR3 );
I_20 : NEG
  Port Map ( IX=>I6, RX=>R6, IY=>NI6, RY=>NR6 );
I_13 : NEG
  Port Map ( IX=>I4, RX=>R4, IY=>NI4, RY=>NR4 );
I_12 : NEG
  Port Map ( IX=>I3, RX=>R3, IY=>NI3, RY=>NR3 );
I_4 : NEG
  Port Map ( IX=>I2, RX=>R2, IY=>NI2, RY=>NR2 );
I_3 : GENCHIRP
  Port Map ( BW=>BW, FREQ=>FREQ, N=>N, PRF=>PRF, PW=>PW,
    SAMP_FREQ=>SAMP_FREQ, I=>I1, R=>R1 );
I_5 : DELAY
Generic Map ( DEL => 25 )
  Port Map ( X=>I1, Y=>DI1 );
I_6 : DELAY
Generic Map ( DEL => 25 )
  Port Map ( X=>R1, Y=>DR1 );
I_18 : DECIMATE
Generic Map ( DEC => DEC )
  Port Map ( IX=>IRH1, N=>N, RX=>RRH1, IY=>IRH, RY=>RRH );
I_26 : DECIMATE
Generic Map ( DEC => DEC )
  Port Map ( IX=>IRV1, N=>N, RX=>RRV1, IY=>IRV, RY=>RRV );
I_21 : COMPTFP
  Port Map ( FREQ=>FREQ, N=>N, SAMP_FREQ=>SAMP_FREQ, I=>I6, R=>R6 );
I_11 : COMPTFP
  Port Map ( FREQ=>FREQ, N=>N, SAMP_FREQ=>SAMP_FREQ, I=>I3, R=>R3 );
I_7 : COMPTFP
  Port Map ( FREQ=>FREQ, N=>N, SAMP_FREQ=>SAMP_FREQ, I=>I2, R=>R2 );
I_22 : COMPMUL
  Port Map ( IIN1=>NI4, IIN2=>I7, RIN1=>NR4, RIN2=>R7, IOUT=>IRV1,
    ROUT=>RRV1 );
I_14 : COMPMUL
  Port Map ( IIN1=>NI4, IIN2=>I5, RIN1=>NR4, RIN2=>R5, IOUT=>IRH1,
    ROUT=>RRH1 );
I_23 : COMPMUL
  Port Map ( IIN1=>DI1, IIN2=>PI6, RIN1=>DR1, RIN2=>PR6, IOUT=>I7,
    ROUT=>R7 );
I_15 : COMPMUL
  Port Map ( IIN1=>DI1, IIN2=>PI3, RIN1=>DR1, RIN2=>PR3, IOUT=>I5,
    ROUT=>R5 );
I_8 : COMPMUL
```

Appendix B

Port Map (IIN1=>NI2, IIN2=>I1, RIN1=>NR2, RIN2=>R1, IOUT=>I4,
ROUT=>R4);

end SCHEMATIC;

configuration CFG_SAR_TB_SCHEMATIC of SAR_TB is

```
for SCHEMATIC
  for I_16, I_25, I_24, I_17: CHANGE
    use configuration WORK.CFG_CHANGE_BEHAVIORAL;
  end for;
  for I_19, I_9: PASS
    use configuration WORK.CFG_PASS_BEHAVIORAL;
  end for;
  for I_20, I_13, I_12, I_4: NEG
    use configuration WORK.CFG_NEG_BEHAVIORAL;
  end for;
  for I_3: GENCHIRP
    use configuration WORK.CFG_GENCHIRP_SCHEMATIC;
  end for;
  for I_5, I_6: DELAY
    use configuration WORK.CFG_DELAY_BEHAVIORAL;
  end for;
  for I_18, I_26: DECIMATE
    use configuration WORK.CFG_DECIMATE_BEHAVIORAL;
  end for;
  for I_21, I_11, I_7: COMPTFP
    use configuration WORK.CFG_COMPTFP_BEHAVIORAL;
  end for;
  for I_22, I_14, I_23, I_15, I_8: COMPMUL
    use configuration WORK.CFG_COMPMUL_BEHAVIORAL;
  end for;
end for;
```

end CFG_SAR_TB_SCHEMATIC;

Procedure based model for IRST Testbench

Package VER3V_PRIM1

```
use work.all;
use STD.TEXTIO.all;
use work.USER_TYPES1.all;
use work.MATH_REAL.all;
```

package VER3V_PRIM1 is

```
procedure OSC22_P ( INIT : In  BIT;
                   PER : In  TIME;
```

Appendix B

```
        CLK : Out BIT );

procedure ENVIRON22_P(DEL :in TIME := 0 NS;
        CLK : In BIT;
        CLUT : Out INT_2D(0 TO 31,0 TO 31) );

procedure TARGET22_P ( DEL : In TIME := 0 NS;
        REG1_RANGE : INTEGER := 5;
        REG2_RANGE : INTEGER := 10;
        REG3_RANGE: INTEGER := 15;
        FRAME_SIZE : INTEGER := 32;
        ANGLE : In INTEGER;
        CLK : In BIT;
        INIT : In BIT;
        MODE : In BIT := '0';
        MODE_MOTION : In INTEGER;
        RUN : In BIT;
        TARG_RANGE : In INTEGER;
        VELOCITY : In INTEGER;
        SEED1 : Inout INTEGER;
        SEED2 : Inout INTEGER;
        X_COORD : Inout INTEGER;
        Y_COORD : Inout INTEGER;
        FRAME : Out INT_2D(0 TO 31,0 TO 31);
        START_FLAG : In BIT);

procedure SENSOR22_P( DEL : In TIME := 0 NS;
        CLK : In BIT;
        CLUT : In INT_2D(0 TO 31,0 TO 31);
        FRAME : In INT_2D(0 TO 31,0 TO 31);
        FRAME_CNT : Inout INTEGER;
        INIT : In BIT;
        IMAGES : Out TEXT;
        IMAGE : Out INT_2D(0 TO 31,0 TO 31));

end VER3V_PRIM1;

package body VER3V_PRIM1 is

-- OSC22_P

procedure OSC22_P ( INIT : In BIT;
        PER : In TIME;
        CLK : Out BIT ) is

begin

while INIT = '1' loop
    CLK := '1';
    wait for PER/2;
```

Appendix B

```
CLK := '0';
wait for PER/2;
end loop;

end OSC22_P;

-- ENVIRON22_P

procedure ENVIRON22_P(DEL :in TIME := 0 NS;
                      CLK : In  BIT;
                      CLUT : Out INT_2D(0 TO 31,0 TO 31) ) is

variable VLINE: LINE;
variable I: INTEGER:= 0;
variable J: INTEGER:= 0;
variable TEMP: INTEGER:= 0;
variable CLUT_CNT : INTEGER:= 0;
variable CLUT_TEMP: INT_2D(0 to 31,0 to 31):= (others => (others => 0));
file INCLUTTER: TEXT is in
"/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/STRUCTURAL/inclutter.txt";

begin

--wait on CLK until CLK = '1';

if not(ENDFILE(INCLUTTER)) then

    READLINE(INCLUTTER,VLINE);
    READ(VLINE, TEMP);

    CLUT_CNT := TEMP;

    if not(CLUT_CNT = 999) then

        for I in 0 to 31 loop
            READLINE(INCLUTTER,VLINE);
            for J in 0 to 31 loop
                READ(VLINE, TEMP);
                CLUT_TEMP(I,J) := TEMP;
            end loop;
        end loop;

    end if;

-- while not(ENDFILE(INCLUTTER)) loop

--    READLINE(INCLUTTER,VLINE);

--    for J in 0 to 31 loop
--        READ(VLINE, TEMP);
```

Appendix B

```
--    CLUT_TEMP(I,J) := TEMP;
-- end loop;

--    I:= I+1;

-- end loop;

CLUT := CLUT_TEMP;

end if;

end ENVIRON22_P;

-- TARGET22_P

procedure TARGET22_P ( DEL : In TIME := 0 NS;
                      REG1_RANGE : INTEGER := 5;
                      REG2_RANGE : INTEGER := 10;
                      REG3_RANGE : INTEGER := 15;
                      FRAME_SIZE : INTEGER := 32;
                      ANGLE : In INTEGER;
                      CLK : In BIT;
                      INIT : In BIT;
                      MODE : In BIT := '0';
                      MODE_MOTION : In INTEGER;
                      RUN : In BIT;
                      TARG_RANGE : In INTEGER;
                      VELOCITY : In INTEGER;
                      SEED1 : Inout INTEGER;
                      SEED2 : Inout INTEGER;
                      X_COORD : Inout INTEGER;
                      Y_COORD : Inout INTEGER;
                      FRAME : Out INT_2D(0 TO 31,0 TO 31);
                      START_FLAG : In BIT) is

variable FRAME_TEMP: INT_2D(0 to 31,0 to 31):= (others => (others => 0));
variable VLINE1: LINE;
variable ANGLE_RAD: REAL;
variable TEMP_ANGLE: INTEGER;
variable I: INTEGER:= 0;
variable I1: INTEGER:= 1;
variable J: INTEGER:= 0;
variable NEW_X_COORD,NEW_Y_COORD:INTEGER:= 0;
variable TEMP1: INTEGER:= 0;
variable TEMP_X,TEMP_Y: INTEGER := 0;
variable TAR_POS: INT_2D(0 to 20,0 to 1):= (others => (others => 0));
variable FLAG1 : BIT := '0';
variable RAND_NO1,RAND_NO2 : REAL;
```

Appendix B

```
file POSITION : TEXT is in "/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/target.in";

begin

-- wait on INIT, CLK, RUN;

if MODE = '0' then

    if INIT = '1' and RUN = '0' then
        FRAME := FRAME_TEMP;
        NEW_X_COORD := X_COORD;
        NEW_Y_COORD := Y_COORD;
    end if;

    if RUN = '1' then
        if CLK = '1' then

            NEW_X_COORD := X_COORD;
            NEW_Y_COORD := Y_COORD;

            FRAME_TEMP(TEMP_X,TEMP_Y) := 0;

            case MODE_MOTION is
                when 1 => TEMP_ANGLE := 0;
                    FLAG1 := '0';
                when 2 => TEMP_ANGLE := 90;
                    FLAG1 := '0';
                when 3 => TEMP_ANGLE := ANGLE;
                    FLAG1 := '0';
                when 4 => UNIFORM(SEED1,SEED1,RAND_NO1);
                    UNIFORM(SEED2,SEED2,RAND_NO2);
                    FLAG1 := '1';
                when others => null;

            end case;

            if ((FLAG1 = '0') and (START_FLAG = '1')) then
                ANGLE_RAD:= MATH_DEG_TO_RAD*REAL(TEMP_ANGLE);
                NEW_X_COORD := NEW_X_COORD +
INTEGER(REAL(VELOCITY)*COS(ANGLE_RAD));
                NEW_Y_COORD := NEW_Y_COORD + INTEGER(REAL(VELOCITY)*SIN(ANGLE_RAD));
            elsif ((FLAG1 = '1') and (START_FLAG = '1')) then
                NEW_X_COORD := INTEGER(31.0*RAND_NO1);
                NEW_Y_COORD := INTEGER(31.0*RAND_NO2);
            end if;

            if ((NEW_X_COORD<32) and (NEW_Y_COORD<32)) then
```

Appendix B

```

    X_COORD := NEW_X_COORD;
    Y_COORD := NEW_Y_COORD;

    TEMP_X := 31-NEW_Y_COORD;
    TEMP_Y := NEW_X_COORD;

    if (15*NEW_Y_COORD <= (-31*NEW_X_COORD+465)) then
        if REG1_RANGE > TARG_RANGE then
            FRAME_TEMP(TEMP_X,TEMP_Y) := 255 ;
        end if;
    elsif ((7*NEW_Y_COORD <= (5*NEW_X_COORD+20)) and (15*NEW_Y_COORD > (-
31*NEW_X_COORD+465))) then
        if REG2_RANGE > TARG_RANGE then
            FRAME_TEMP(TEMP_X,TEMP_Y) := 255 ;
        end if;
    else
        if REG3_RANGE > TARG_RANGE then
            FRAME_TEMP(TEMP_X,TEMP_Y) := 255 ;
        end if;
    end if;

    end if;

    FRAME := FRAME_TEMP;

end if;
end if;

elsif MODE = '1' then

    if INIT = '1' and RUN = '0' then

        while not(ENDFILE(POSITION)) loop

            READLINE(POSITION,VLINE1);

            for J in 0 to 1 loop
                READ(VLINE1, TEMP1);
                TAR_POS(I,J):= TEMP1;
            end loop;

            I:= I+1;

        end loop;

        NEW_X_COORD := TAR_POS(0,0);
        NEW_Y_COORD := TAR_POS(0,1);
    end if;
end if;
```

Appendix B

```
end if;

if RUN = '1' then
  if CLK = '1' then

    FRAME_TEMP(TEMP_X,TEMP_Y) := 0;

    if ((NEW_X_COORD<32) and (NEW_Y_COORD<32)) then

      X_COORD := NEW_X_COORD;
      Y_COORD := NEW_Y_COORD;

      TEMP_X:= 31-NEW_Y_COORD;
      TEMP_Y := NEW_X_COORD;

      if (15*NEW_Y_COORD <= (-31*NEW_X_COORD+465)) then
        if REG1_RANGE > TARG_RANGE then
          FRAME_TEMP(TEMP_X,TEMP_Y) := 255 ;
        end if;
      elsif ((7*NEW_Y_COORD <= (5*NEW_X_COORD+20)) and (15*NEW_Y_COORD > (-
31*NEW_X_COORD+465))) then
        if REG2_RANGE > TARG_RANGE then
          FRAME_TEMP(TEMP_X,TEMP_Y) := 255 ;
        end if;
      else
        if REG3_RANGE > TARG_RANGE then
          FRAME_TEMP(TEMP_X,TEMP_Y) := 255 ;
        end if;
      end if;

    end if;

    if not((TAR_POS(I1,0)=0) and (TAR_POS(I1,1)=0)) then
      NEW_X_COORD := TAR_POS(I1,0);
      NEW_Y_COORD := TAR_POS(I1,1);
      I1 := I1+1;
    end if;

    FRAME := FRAME_TEMP;

  end if;
end if;

end if;

end TARGET22_P;
```

Appendix B

-- SENSOR22_P

```
procedure SENSOR22_P( DEL : In TIME := 0 NS;
                     CLK : In  BIT;
                     CLUT : In  INT_2D(0 TO 31,0 TO 31);
                     FRAME : In  INT_2D(0 TO 31,0 TO 31);
                     FRAME_CNT : Inout INTEGER;
                     INIT : In  BIT;
                     IMAGES : Out TEXT;
                     IMAGE : Out  INT_2D(0 TO 31,0 TO 31)) is
```

```
variable I: INTEGER:= 0;
variable J: INTEGER:= 0;
-- variable COUNT: INTEGER:= 1;
variable COUNT1: INTEGER:= 999;
variable IMAGE_TEMP: INT_2D(0 to 31,0 to 31):= (others => (others => 0));
variable VLINE2: LINE;
variable SPACE: CHARACTER:= ' ';
```

begin

--wait on INIT, CLK until CLK = '1';

if INIT = '1' then

--wait for DEL;

```
for I in 0 to 31 loop
  for J in 0 to 31 loop
    if ((CLUT(I,J)=255) or (FRAME(I,J)=255)) then
      IMAGE_TEMP(I,J) := 255;
    else
      IMAGE_TEMP(I,J) := CLUT(I,J) + FRAME(I,J);
    end if;
    --IMAGE(I,J) := IMAGE_TEMP(I,J);
  end loop;
end loop;
```

IMAGE := IMAGE_TEMP;

```
WRITE(VLINE2,FRAME_CNT);
WRITELINE(IMAGES,VLINE2);
```

for I in 0 to 31 loop

for J in 0 to 31 loop

Appendix B

```
        WRITE(VLINE2,IMAGE_TEMP(I,J));
        WRITE(VLINE2,SPACE);
    end loop;

    WRITELINE(IMAGES,VLINE2);

end loop;

FRAME_CNT:= FRAME_CNT + 1;

end if;

end SENSOR22_P;

end VER3V_PRIM1;
```

Procedure based Testbench for IRST

```
use work.all;
use STD.TEXTIO.all;
use work.USER_TYPES1.all;
use work.MATH_REAL.all;
use work.VER3V_PRIM1.all;

entity TB22v1 is
    Port ( ANGLE : In  INTEGER;
          INIT : In  BIT;
          MODE : In  BIT;
          MODE_MOTION : In  INTEGER;
          DEL : In  TIME;
          PER : In  TIME;
          TARG_RANGE : In  INTEGER;
          RUN : In  BIT;
          VELOCITY : In  INTEGER;
          X_COORD : Inout INTEGER;
          Y_COORD : Inout INTEGER;
          IMAGE : Out  INT_2D(0 TO 31,0 TO 31) );
end TB22v1;

architecture BHVR of TB22v1 is

    signal    CLK : BIT;

begin

    OSC: process

        variable CLKV : BIT;
```

Appendix B

```
begin

-- OSC22(INIT,PER,CLKV);

while INIT = '1' loop
CLK <= '1';
wait for PER/2;
CLK <= '0';
wait for PER/2;
end loop;

end process;

MAIN: process

variable FRAME : INT_2D(0 TO 31,0 TO 31) := (OTHERS => (OTHERS => 0));
variable CLUT : INT_2D(0 TO 31,0 TO 31) := (OTHERS => (OTHERS => 0));
variable IMAGE1 : INT_2D(0 TO 31,0 TO 31) := (OTHERS => (OTHERS => 0));
variable X_COORD1,Y_COORD1 : INTEGER := 0;
variable FRAME_CNT : INTEGER := 1;
variable REG1_RANGE : INTEGER := 5;
variable REG2_RANGE : INTEGER := 10;
variable REG3_RANGE : INTEGER := 15;
variable FRAME_SIZE : INTEGER := 32;
variable START_FLAG : BIT := '0';
variable SEED1 : INTEGER := 2000;
variable SEED2 : INTEGER := 5000;
file IMAGES : TEXT is out "/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/images.txt";

begin

wait on CLK until CLK = '1';

ENVIRON22_P(DEL,CLK,CLUT);

X_COORD1 := X_COORD;

Y_COORD1 := Y_COORD;

TARGET22_P(DEL,REG1_RANGE,REG2_RANGE,REG3_RANGE,FRAME_SIZE,ANGLE,CLK,INIT,
MODE,MODE_MOTION,RUN,TARG_RANGE,VELOCITY,SEED1,SEED2,X_COORD1,Y_COORD1,
FRAME,START_FLAG);

if (RUN = '1') then
START_FLAG := '1';
end if;

X_COORD <= X_COORD1;

Y_COORD <= Y_COORD1;
```

Appendix B

```
SENSOR22_P(DEL,CLK,CLUT,FRAME,FRAME_CNT,INIT,IMAGES,IMAGE1);  
  
IMAGE <= IMAGE1;  
  
end process;  
  
end BHVR;
```

Appendix C: Code Listing for the Intelligent User Interface

("inter.c")

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <strings.h>
#include <math.h>

# define LINELEN 20

main ()
{
    static char dum_ch[10];
    int alg_choice = 0;

    /* Obtaining inputs */

START :    printf("\n\n\t\t\t\t\t SIGNAL PROCESSING TEST BENCH \n");
           printf("\t\t\t\t\t ----- \n\n\n");

           printf("Choose the application for which the test bench needs to be created : \n\n");
           printf("\n1 -> 2D Image Processing Algorithm\n\n");
           printf("2 -> SAR Algorithm\n\n");
           printf("3 -> Quit\n\n");
           printf("Enter Choice : ");
```

Appendix C

```
scanf("%s",dum_ch);
printf("\n\n");

alg_choice = atoi(dum_ch);

/* Re-direction to application */

if (alg_choice == 1)
    call_irstalg();
else if (alg_choice == 2)
{
    system("xu_exec");
}
else if (alg_choice == 3)
    exit(0);
else
{
    printf("\n\nSorry. Wrong Input. Try again.\n\n");
    goto START;
}

goto START;

}

/*****
Function call_irstalg()
*****/

call_irstalg()
{
    int driven_choice,model_choice,proc_choice,out_flag = 0;
    FILE *scl_file;
    static char dum_ch[10],dr_ch[10];

START1 : printf("\f");
printf("\n\n\n");
printf("\t\t2D Image Processing Test Bench\n");
printf("\t\t-- ----- ---- -----\n\n");

printf("Do you want the test bench to be ....\n\n");
printf("1 -> Requirements Driven\n\n");
printf("2 -> User Driven\n\n");
printf("3 -> Previous Menu\n\n");
printf("Enter choice : ");
```

Appendix C

```
scanf("%s",dr_ch);
printf("\n\n");

driven_choice = atoi(dr_ch);

if ( driven_choice == 3)
{
    return;
}

printf("\f\n\nWhich of the following types of test benches do you want to run ? \n\n");
printf("1 -> Behavioral Model\n\n");
printf("2 -> Structural Model\n\n");
printf("3 -> Previous Menu\n\n");
printf("Enter Choice : ");
scanf("%s",dum_ch);

model_choice = atoi(dum_ch);

scl_file = fopen("/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/tb1.con","w");

/* Re-direction to application */

if (model_choice == 1 || model_choice == 2)
    out_flag = form_con(scl_file,model_choice,driven_choice);
else if (model_choice == 3)
    goto START1;
else
{
    printf("\n\nSorry. Wrong Input. Try again.\n\n");
    goto START1;
}

fclose(scl_file);

if (out_flag == 1)
    goto START1;

PROC : printf("\n\nChoose between ..... \n\n");
printf("1 -> On-line Processing\n\n");
printf("2 -> Post Processing\n\n");
printf("Enter choice : ");
scanf("%s",dum_ch);
printf("\n\n");

proc_choice = atoi(dum_ch);
```

Appendix C

```
if (proc_choice != 2 && proc_choice != 1)
{
    printf("\n\nSorry. Invalid Choice. Try again.\n\n");
    goto PROC;
}

printf("\n\nCreating SCL File ..... \n\n");
printf("\n\nSimulating ..... \n\n");

if (model_choice == 1 && proc_choice == 1)
{
    system("cat
/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/BEHAVIORAL/tb1_temp_beh1.con >>
/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/tb1.con");
}
else if (model_choice == 2 && proc_choice == 1)
{
    system("cat
/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/STRUCTURAL/tb1_temp_str1.con >>
/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/tb1.con");
}
else if (model_choice == 1 && proc_choice == 2)
{
    system("cat
/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/BEHAVIORAL/tb1_temp_beh2.con >>
/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/tb1.con");
}
else if (model_choice == 2 && proc_choice == 2)
{
    system("cat
/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/STRUCTURAL/tb1_temp_str2.con >>
/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/tb1.con");
}

if (model_choice == 1 && proc_choice == 2)
{
    system("vhdlan -nc
/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/BEHAVIORAL/first_latest.vhd");
    system("xterm -fn fixed -fg white -bg black -geometry 95x9+235+750 -T
TEST_BENCH_SIMULATION -e vhdlsim -t ns -nc -i
/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/tb1.con SPTB2 ");
    printf("\n\nPost processing data and plotting ..... \n\n");
    system("matlab < /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/plotter_bmp.m
");
}
else if (model_choice == 1 && proc_choice == 1)
```

Appendix C

```
{
    system("xterm -fn fixed -fg white -bg black -geometry 95x9+235+750 -T
TEST_BENCH_SIMULATION -e vhdlsim -t ns -nc -i
/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/tb1.con SPTB2 ");
}
else if (model_choice == 2 && proc_choice == 1)
{
    system("xterm -fn fixed -fg white -bg black -geometry 95x9+235+750 -T
TEST_BENCH_SIMULATION -e vhdlsim -t ns -nc -i
/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/tb1.con CFG_TB22_SCHEMATIC ");
}
else if (model_choice == 2 && proc_choice == 2)
{
    system("xterm -fn fixed -fg white -bg black -geometry 95x9+235+750 -T
TEST_BENCH_SIMULATION -e vhdlsim -t ns -nc -i
/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/tb1.con CFG_TB22_SCHEMATIC ");

    printf("\n\nPost processing data and plotting ..... \n\n");
    system("matlab < /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/plotter_bmp.m
");
}

return;

}

/*****
Function Form_con()
*****/

form_con(fil,model_choice,driven_choice)
FILE *fil;
int model_choice,driven_choice;
{

    char *pos_x;
    char *pos_y;
    char *veloc;
    char *mod_motion_ptr;
    char *angle;
    char *start_time;
    char *end_time;
    char *clk_per;
    static char del[100];
    static char x[100] = "assign ";
    static char x1_beh[] = " XCOORD";
    static char x1_str[] = " X_COORD";
    static char y[100] = "assign ";
    static char y1_beh[] = " YCOORD";
```

Appendix C

```
static char y1_str[] = " Y_COORD";
static char v[100] = "assign ";
static char v1[100] = " VELOCITY";
static char mod_motion_1[100] = "assign ";
static char mod_motion_2[100] = " MODE_MOTION";
static char a[100] = "assign ";
static char a1[] = " ANGLE";
static char reg_beh[] = "cd SPTB2";
static char reg_str[] = "cd TB22";
static char init_beh[] = "assign ('1') di1INIT";
static char init_str[] = "assign ('1') INIT";
static char trig[100] = "assign -after ";
static char trig1_beh[] = " ('1') TRIG";
static char trig1_str[] = " ('1') RUN";
static char stop[100] = "assign -after ";
static char stop1_beh[] = " ('0') TRIG";
static char stop1_str[] = " ('0') RUN";
static char stop2_str[] = " ('0') INIT";
static char mode_beh[] = "assign (1) MODE";
static char mode_str[] = "assign ('1') MODE";
static char period[100] = "assign ";
static char period1[] = " ns) PER";
static char set_per[] = "set PERIOD ";

int temp_targ_mov;

int init_x_file,init_y_file,calc_x_file,calc_y_file,veloc_file,angle_file;
int choice,motion_choice,mode_choice,file_choice, number =
0,motion_file_choice,i,data_check_flag;
int
limit_check_flag1,limit_check_flag2,limit_check_flag3,limit_check_flag4,integer_check_flag1,integer_c
heck_flag2,integer_check_flag3,integer_check_flag4,integer_check_flag5,integer_check_flag6;
char
*tarfil_nam,*buffer,*dummy,*trig_sig,*stop_sig,*period_ptr,*set_per_ptr,*stop_dup_ptr,*x_ptr,*y_ptr,*a
_ptr,*v_ptr;
FILE *tar_file,*tar_file_created,*tar_act_file,*template_file;
static char dum_ch1[10], dum_ch2[10] , mod_motion_ch[10] ;
int temp_vcos,temp_vsin,ivcos,ivsin;
double cos_angle,sin_angle;
float vcos,vsin;
float temp_angle_file;

buffer = (char *) malloc(30 * sizeof(char));
tarfil_nam = (char *) malloc(30 * sizeof(char));
dummy = (char *) malloc(30 * sizeof(char));

pos_x = (char *) calloc(1,100 * sizeof(char));
pos_y = (char *) calloc(1,100 * sizeof(char));
veloc = (char *) calloc(1,100 * sizeof(char));
mod_motion_ptr = (char *) calloc(1,100 * sizeof(char));
```

Appendix C

```
angle = (char *) calloc(1,100 * sizeof(char));
start_time = (char *) calloc(1,100 * sizeof(char));
end_time = (char *) calloc(1,100 * sizeof(char));
trig_sig = (char *) calloc(1,100*sizeof(char));
stop_sig = (char *) calloc(1,100*sizeof(char));
period_ptr = (char *) calloc(1,100 * sizeof(char));
stop_dup_ptr = (char *) calloc(1,100 * sizeof(char));
set_per_ptr = (char *) calloc(1,100 * sizeof(char));
x_ptr = (char *) calloc(1,100 * sizeof(char));
y_ptr = (char *) calloc(1,100 * sizeof(char));
a_ptr = (char *) calloc(1,100 * sizeof(char));
v_ptr = (char *) calloc(1,100 * sizeof(char));

if (driven_choice == 1)
{

temp_targ_mov = req_file_formcon();
mode_choice = 2;
sprintf(veloc,"%d",temp_targ_mov);

}
else
{
printf("\f");
printf("\n\n");
}

START2 :    printf("\n\nModes of Operation :\n");
            printf("----- -- -\n\n");
            printf("\n1 -> File I/O for Target Info\n");
            printf("\n2 -> Target Info based on Initial Position\n");
            printf("\n3 -> Previous Menu\n");
            printf("Enter choice      : ");
            scanf("%s",dum_ch1);

            mode_choice = atoi(dum_ch1);

            printf("\n\n");

        }

if (mode_choice != 1 && mode_choice != 2 && mode_choice != 3)
{
            printf("\n\nSorry. Wrong Input. Try again.\n\n");
            goto START2;
}

if (mode_choice == 3)
{
            return(1);
}
```

Appendix C

```
    }
    else if (mode_choice == 1)
    {

START3 :    printf("\n\nOptions : \n\n");
            printf("1 -> Use already created file \n\n");
            printf("2 -> Create a new file \n\n");
            printf("3 -> Previous Menu\n\n");
            printf("Enter choice : ");
            scanf("%s",dum_ch2);
            printf("\n\n");

            file_choice = atoi(dum_ch2);

            if (file_choice != 1 && file_choice != 2 && file_choice != 3 )
            {
                printf("\n\nSorry. Wrong Input. Try again.\n\n");
                goto START3;
            }

            if (file_choice == 3)
                goto START2;
            else if (file_choice == 1)
            {

                tar_file =
fopen("/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/target.in","w");

                gets(dummy);

START4 :    tarfil_nam = (char *) calloc(1,30 * sizeof(char));
            printf("Enter name of file that has been already created (give absolute path) :
");

            scanf("%s",tarfil_nam);

            if ((tar_file_created = fopen(tarfil_nam,"r")) == NULL)
            {
                printf("\n\nSorry. File does not exist. \n\n");
                printf("1 -> Try again\n\n");
                printf("2 -> Previous menu\n\n");
                printf("Enter choice : ");
                scanf("%s",dum_ch2);
                printf("\n\n");

START5 :    choice = atoi(dum_ch2);

                if (choice == 1)
                    goto START4;
                else if (choice == 2)
                    goto START3;
                else
```

Appendix C

```
        {
            printf("Sorry. Invalid choice. Try again.\n\n");
            goto START5;
        }
    }

    while (!feof(tar_file_created))
    {
        fgets(buffer,LINELEN,tar_file_created);
        if (!feof(tar_file_created))
            fputs(buffer,tar_file);
    }

    fclose(tar_file_created);
    fclose(tar_file);

}
else if (file_choice == 2)
{
    printf("\n\nEnter name of the file to create (give absolute path) : ");
    gets(dummy);
    gets(tarfil_nam);

    printf("\n\nEnter number of target points that you want to create : ");
    scanf("%d",&number);
```

FILE_MOTION :

```
        printf("\n\nMovement options : \n\n");
        printf("1 -> Horizontal Motion\n\n");
        printf("2 -> Vertical Motion\n\n");
        printf("3 -> Climb\n\n");
        printf("4 -> Random Walk\n\n");
        printf("5 -> User-defined points\n\n");
        printf("6 -> Previous Menu\n\n");
        printf("Enter choice : ");
        scanf("%d",&motion_file_choice);

        if (motion_file_choice == 6)
            goto START3;

        tar_file = fopen(tarfil_nam,"w");
        tar_act_file = fopen("/home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/target.in","w");

        buffer = (char *) malloc(30 * sizeof(char));

        if (motion_file_choice != 5)
        {
```

Appendix C

```
printf("\nEnter initial X co-ordinate : ");
scanf("%d",&init_x_file);
printf("\nEnter initial Y co-ordinate : ");
scanf("%d",&init_y_file);

if (init_x_file > 31 || init_y_file > 31)
{
printf("\n\nSorry. Invalid data.\n\n");
goto FILE_MOTION;
}

fprintf(tar_file,"%d %d\n",init_x_file,init_y_file);
fprintf(tar_act_file,"%d %d\n",init_x_file,init_y_file);

if (motion_file_choice != 4 )
{
printf("\nEnter velocity of motion (0 to 5) : ");
scanf("%d",&veloc_file);

if (veloc_file > 5)
{
printf("\n\nSorry. Invalid data. \n\n");
goto FILE_MOTION;
}
}

if (motion_file_choice == 1)
{
for (i=1; i <= number-1;i++)
{
calc_x_file = init_x_file + veloc_file;
fprintf(tar_file,"%d %d\n",calc_x_file,init_y_file);
fprintf(tar_act_file,"%d
%d\n",calc_x_file,init_y_file);

init_x_file = calc_x_file;
}
}
else if (motion_file_choice == 2)
{
for (i=1; i <= number-1;i++)
{
calc_y_file = init_y_file + veloc_file;
fprintf(tar_file,"%d %d\n",init_x_file,calc_y_file);
fprintf(tar_act_file,"%d
%d\n",init_x_file,calc_y_file);

init_y_file = calc_y_file;
}
}
else if (motion_file_choice == 3)
{
```

Appendix C

```
printf("\nEnter angle of movement in degrees (0 to 90) : ");
scanf("%d",&angle_file);

if (angle_file > 90)
{
    printf("\n\nSorry. Invalid data.\n\n");
    goto FILE_MOTION;
}

temp_angle_file = angle_file/180.0 * M_PI;
cos_angle = cos(temp_angle_file);
sin_angle = sin(temp_angle_file);

vcos = veloc_file * cos_angle;
vsin = veloc_file * sin_angle;

temp_vcos = vcos;
temp_vsin = vsin;

if (vcos - temp_vcos >= 0.5)
{
    ivcos = temp_vcos + 1;
}
else
    ivcos = temp_vcos;
if (vsin - temp_vsin >= 0.5)
    ivsin = temp_vsin + 1;
else
    ivsin = temp_vsin;

for (i = 1; i <= number-1; i++)
{

    calc_x_file = init_x_file + ivcos;
    calc_y_file = init_y_file + ivsin;
    init_x_file = calc_x_file;
    init_y_file = calc_y_file;
    fprintf(tar_file,"%d %d\n",calc_x_file,calc_y_file);
    fprintf(tar_act_file,"%d
%d\n",calc_x_file,calc_y_file);

}

}
else if (motion_file_choice == 4)
{

    for (i = 1; i <= number-1; i++)
    {
        calc_x_file = rand()/2147483647.0*31.0;
        calc_y_file = rand()/2147483647.0*31.0;
        fprintf(tar_file,"%d %d\n",calc_x_file,calc_y_file);
```

Appendix C

```

                                                                    fprintf(tar_act_file,"%d
%d\n",calc_x_file,calc_y_file);
                                                                    }
                                                                    }
                                                                    }
                                                                    else
                                                                    {

AGAIN :                                                                    printf("\n\nEnter the x-coordinates and y-coordinates in 'xx yy'
format\n\n");

                                                                    gets(dummy);

                                                                    for (i=0;i<number;i++)
                                                                    {
                                                                    printf("Point # %d : ",i+1);
                                                                    gets(buffer);

                                                                    data_check_flag = data_check(buffer);

                                                                    if (data_check_flag == 1)
                                                                    {
                                                                    printf("\nSorry. Data format is not correct. Enter the X and
Y co-ordinates SEPERATED by a BLANK.\n\n");
                                                                    goto AGAIN;
                                                                    }

                                                                    printf("\n");
                                                                    fputs(buffer,tar_file);
                                                                    fputs(buffer,tar_act_file);
                                                                    fputs("\n",tar_file);
                                                                    fputs("\n",tar_act_file);
                                                                    }
                                                                    }

                                                                    fclose(tar_file);
                                                                    fclose(tar_act_file);
                                                                    }

/* Target file has been created. Now get required inputs */

AGAIN_1 :    start_time = (char *) calloc(1,100 * sizeof(char));
                                                                    printf("\n\nEnter time in NS at which target update starts : ");
                                                                    scanf("%s",start_time);
                                                                    integer_check_flag4 = integer_check(start_time);
                                                                    if (integer_check_flag4 == 1)
                                                                    {
                                                                    printf("\nSorry. Invalid data. Try again.\n");
                                                                    goto AGAIN_1;
                                                                    }
                                                                    }
```

```

    }

AGAIN_2 :      end_time = (char *) calloc(1,100 * sizeof(char));
               printf("\nEnter time in NS at which target update stops   : ");
               scanf("%s",end_time);
               integer_check_flag5 = integer_check(end_time);
               if (integer_check_flag5 == 1)
               {
                 printf("\nSorry. Invalid data. Try again.\n");
                 goto AGAIN_2;
               }

AGAIN_3 :      clk_per = (char *) calloc(1,100 * sizeof(char));
               printf("\nEnter time in NS between updates in target position : ");
               scanf("%s",clk_per);
               integer_check_flag6 = integer_check(clk_per);
               if (integer_check_flag6 == 1)
               {
                 printf("\nSorry. Invalid data. Try again.\n");
                 goto AGAIN_3;
               }

}
else if (mode_choice == 2)
{

    if (driven_choice != 1)
        printf("\n\t\t\t\t\tTARGET DETAILS\n\n");

AGAIN1 :      pos_x = (char *) calloc(1,100 * sizeof(char));
               printf("\nEnter X-Coordinate   (From 0 to 32)           : ");
               scanf("%s",pos_x);
               limit_check_flag1 = limit_check(pos_x,'3','2','9');
               if (limit_check_flag1 == 1)
               {
                 printf("\nSorry. Invalid data. Try again.\n");
                 goto AGAIN1;
               }

AGAIN2 :      pos_y = (char *) calloc(1,100 * sizeof(char));
               printf("\nEnter Y-Coordinate   (From 0 to 32)           : ");
               scanf("%s",pos_y);
               limit_check_flag2 = limit_check(pos_y,'3','2','9');
               if (limit_check_flag2 == 1)
               {
                 printf("\nSorry. Invalid data. Try again.\n");
                 goto AGAIN2;
               }

```

Appendix C

```
        }

    if (driven_choice == 1)
    {
        motion_choice = 3;
        sprintf(mod_motion_ch,"%d",motion_choice);
        goto AGAIN4;
    }

/* changed menu */

    printf("\nChoose mode of motion :\n\n");
    printf("1 -> Horizontal\n\n");
    printf("2 -> Vertical\n\n");
    printf("3 -> Climb\n\n");
    printf("4 -> Random Walk\n\n");
    printf("5 -> Previous menu\n\n");
    printf("Enter choice : ");
    scanf("%s",mod_motion_ch);
    printf("\n\n");

    motion_choice = atoi(mod_motion_ch);

    if (motion_choice == 5)
    {

        goto START2;

    }

    if (motion_choice != 4)
    {
AGAIN3 :    veloc = (char *) calloc(1,100 * sizeof(char));
        printf("\nEnter Velocity      (From 0 to 05)      : ");
        scanf("%s",veloc);
        limit_check_flag3 = limit_check(veloc,'0','5','5');
        if (limit_check_flag3 == 1)
        {
            printf("\nSorry. Invalid data. Try again.\n");
            goto AGAIN3;
        }
    }

    if (motion_choice == 3)
    {
AGAIN4 :    angle = (char *) calloc(1,100 * sizeof(char));
```

Appendix C

```
printf("\nEnter angle of movement (From 0 to 90 degrees)      : ");
scanf("%s",angle);
limit_check_flag4 = limit_check(angle,'9','0','9');
if (limit_check_flag4 == 1)
{
    printf("\nSorry. Invalid data. Try again.\n");
    goto AGAIN4;
}

}

AGAIN5 :    start_time = (char *) calloc(1,100 * sizeof(char));
printf("\nEnter time in NS at which target update starts      : ");
scanf("%s",start_time);
integer_check_flag1 = integer_check(start_time);
if (integer_check_flag1 == 1)
{
    printf("\nSorry. Invalid data. Try again.\n");
    goto AGAIN5;
}

AGAIN6 :    end_time = (char *) calloc(1,100 * sizeof(char));
printf("\nEnter time in NS at which target update stops      : ");
scanf("%s",end_time);
integer_check_flag2 = integer_check(end_time);
if (integer_check_flag2 == 1)
{
    printf("\nSorry. Invalid data. Try again.\n");
    goto AGAIN6;
}

AGAIN7 :    clk_per = (char *) calloc(1,100 * sizeof(char));
printf("\nEnter time in NS between updates in target position : ");
scanf("%s",clk_per);
integer_check_flag3 = integer_check(clk_per);
if (integer_check_flag3 == 1)
{
    printf("\nSorry. Invalid data. Try again.\n");
    goto AGAIN7;
}

}

trig_sig = (char *) calloc(1,100 * sizeof(char));
stop_sig = (char *) calloc(1,100 * sizeof(char));
period_ptr = (char *) calloc(1,100 * sizeof(char));
set_per_ptr = (char *) calloc(1,100 * sizeof(char));
```

Appendix C

```
stop_dup_ptr = (char *) calloc(1,100 * sizeof(char));
x_ptr = (char *) calloc(1,100 * sizeof(char));
y_ptr = (char *) calloc(1,100 * sizeof(char));
a_ptr = (char *) calloc(1,100 * sizeof(char));
v_ptr = (char *) calloc(1,100 * sizeof(char));
```

```
strcat(period_ptr,period);
strcat(period_ptr,clk_per);
strcat(period_ptr,period1);
```

```
strcat(set_per_ptr,set_per);
strcat(set_per_ptr,clk_per);
```

```
if (model_choice == 1)
{
```

```
    strcat(trig_sig, trig);
```

```
    strcat(trig_sig,start_time);
    strcat(trig_sig,trig1_beh);
```

```
    strcat(stop_sig,stop);
    strcat(stop_sig,end_time);
    strcat(stop_sig,stop1_beh);
```

```
    }
else if (model_choice == 2)
{
```

```
    strcat(trig_sig, trig);
```

```
    strcat(trig_sig,start_time);
    strcat(trig_sig,trig1_str);
```

```
    strcat(stop_sig,stop);
    strcat(stop_sig,end_time);
```

```
    strcat(stop_dup_ptr,stop_sig);
```

```
    strcat(stop_sig,stop1_str);
```

```
    strcat(stop_dup_ptr,stop2_str);
```

```
}
```

```
if (mode_choice == 1)
{
```

```
    if (model_choice == 1)
```

Appendix C

```
fprintf(fil, "%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n", reg_beh, init_beh, trig_sig, stop_sig, mode_beh, period_ptr, set_per_ptr);
    else if (model_choice == 2)

        fprintf(fil, "%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n", reg_str, init_str, trig_sig, stop_sig, stop_dup_ptr, mode_str, period_ptr, set_per_ptr);

    }
    else if (mode_choice == 2)
    {

        strcat(v_ptr, v);
        strcat(v_ptr, veloc);
        strcat(v_ptr, v1);

        strcat(mod_motion_ptr, mod_motion_1);
        strcat(mod_motion_ptr, mod_motion_ch);
        strcat(mod_motion_ptr, mod_motion_2);

        strcat(a_ptr, a);
        strcat(a_ptr, angle);
        strcat(a_ptr, a1);

        if (model_choice == 1)
        {

            strcat(x_ptr, x);
            strcat(x_ptr, pos_x);
            strcat(x_ptr, x1_beh);

            strcat(y_ptr, y);
            strcat(y_ptr, pos_y);
            strcat(y_ptr, y1_beh);

            /* Writing to file */

            if (motion_choice == 3)
            {

                fprintf(fil, "%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n", reg_beh, x_ptr, y_ptr, v_ptr, mod_motion_ptr, a_ptr, init_beh, trig_sig, stop_sig, period_ptr, set_per_ptr);
            }
            else if (motion_choice != 4)
            {

                fprintf(fil, "%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n", reg_beh, x_ptr, y_ptr, v_ptr, mod_motion_ptr, init_beh, trig_sig, stop_sig, period_ptr, set_per_ptr);
            }
        }
    }
}
```

Appendix C

```
        }
        else
        {

            fprintf(fil,"%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n",reg_beh,x_ptr,y_ptr,mod_motion_ptr,init_beh,
            trig_sig,stop_sig,period_ptr,set_per_ptr);

        }

        else if (model_choice == 2)
        {

            strcat(x_ptr,x);
            strcat(x_ptr,pos_x);
            strcat(x_ptr,xl_str);

            strcat(y_ptr,y);
            strcat(y_ptr,pos_y);
            strcat(y_ptr,yl_str);

            /* Writing to file */

            if (motion_choice == 3)
            {

                fprintf(fil,"%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n",reg_str,x_ptr,y_ptr,v_ptr,mod_motion_ptr,a_ptr,init_str,
                trig_sig,stop_sig,stop_dup_ptr,period_ptr,set_per_ptr);

            }

            else if (motion_choice != 4)
            {

                fprintf(fil,"%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n",reg_str,x_ptr,y_ptr,v_ptr,mod_motion_ptr,init_str,
                trig_sig,stop_sig,period_ptr,set_per_ptr);

            }

            else
            {

                fprintf(fil,"%s\n%s\n%s\n%s\n%s\n%s\n%s\n%s\n",reg_str,x_ptr,y_ptr,mod_motion_ptr,init_str,
                trig_sig,stop_sig,period_ptr,set_per_ptr);

            }

        }

    }
```

Appendix C

```
        return(0);

    }

/*****
Function data_check()
*****/

data_check(buff)
char *buff;

{
    static char buff_arr[10];
    int i;

    strcpy(buff_arr,buff);

    if (strlen(buff) != 5)
        return(1);
    else
    {
        if (buff_arr[2] != ' ')
            return(1);
        else
        {
            for (i=0;i<=4;i++)
            {
                if (i != 2)
                    if (buff_arr[i] < '0' || buff_arr[i] > '9')
                        return(1);
            }
        }
    }

    return(0);
}
```

Appendix C

```

/*****
                                Function limit_check()
*****/

limit_check(arr,first,second,single_bound)
char *arr;
char first;
char second;
char single_bound;
{

static char arr_dum[100];

    strcpy(arr_dum,arr);

    if (strlen(arr_dum) != 2 && strlen(arr_dum) != 1)
        return(1);

    if (strlen(arr_dum) == 2)
    {
        if (arr_dum[0] > first || arr_dum[0] < '0')
            return(1);

        if (arr_dum[0] == first && (arr_dum[1] > second || arr_dum[1] < '0'))
            return(1);
        else if (arr_dum[1] < '0' || arr_dum[1] > '9')
            return(1);
    }

    if (strlen(arr_dum) == 1)
    {
        if (arr_dum[0] < '0' || arr_dum[0] > single_bound)
            return(1);
    }

    return(0);
}

/*****
                                Function integer_check()
*****/

integer_check(arr)
char *arr;
{

int i;
static char arr_dum[100];
```

Appendix C

```
strcpy(arr_dum,arr);

for(i=0;i<strlen(arr_dum);i++)
{
    if (arr_dum[i] < '0' || arr_dum[i] > '9')
        return(1);
}

return(0);
}

/*****
Function req_file_formcon()
*****/

int req_file_formcon()
{
char *reqfil_nam;
char *dummy;
char *platform_type,*sensor_res;
char c1,c2,dc;
FILE *req_file;
float targ_speed;
int fov,range,revisit_per,bars_perframe;
int platform_vel, target_vel,platform_disp,target_disp;
float clutter_range,target_range;
double clutter_motion, target_motion;
int targ_movement, clutter_movement;
char *veloc;

int e6 = 448, f6 = 256, g6 = 192, d6 = 320;
long float one_by_j12;
static char hi[] = "HI", lo[] = "LOW";

    veloc = (char *) calloc(1,100 * sizeof(char));
    reqfil_nam = (char *) calloc(30,sizeof(char));
    dummy = (char *) calloc(30,sizeof(char));
    platform_type = (char *) calloc(10,sizeof(char));
    sensor_res = (char *) calloc(10,sizeof(char));

    printf("\n\nEnter the name of requirements file (full path) : ");
    gets(dummy);
    gets(reqfil_nam);
```

Appendix C

```
if ((req_file = fopen(reqfil_nam,"r")) == NULL)
{
    printf("\nSorry. Requirements File does not exist. \n");
    printf("Do you want to create one ? (y or n) : ");
    getchar(c1);

    if (c1 == 'n')
        exit(0);
    else
    {
        form_req_file(reqfil_nam);
        req_file = fopen(reqfil_nam,"r");
    }
}

printf("\n\nHere are the independent parameters given in the requirements file :\n\n");
fscanf(req_file,"%f\n%s\n%s\n%d\n%d\n%d\n%d",&targ_speed,platform_type,sensor_res,&fov,
&range,&revisit_per,&bars_perframe);

printf("Target Speed\t\t: %3.1f Mach\nPlatform Type\t\t: %s\nSensor Resolution\t:
%s\nFOV\t\t\t: %d degrees\nRange\t\t\t: %d miles\nRevisit Period\t\t: %d sec\nBars Per Frame\t\t:
%d\n\n",targ_speed,platform_type,sensor_res,fov,range,revisit_per,bars_perframe);

printf("\nDo you want to change the parameters ? (y or n) : ");
fflush(stdin);
scanf("%c",&c2);

if (c2 == 'y')
{
    fclose(req_file);
    form_req_file(reqfil_nam);
    req_file = fopen(reqfil_nam,"r");
}
else
{
    fclose(req_file);
    req_file = fopen(reqfil_nam,"r");
}

platform_type = (char *) calloc(10,sizeof(char));
sensor_res = (char *) calloc(10,sizeof(char));

fflush(req_file);
fscanf(req_file,"%f\n%s\n%s\n%d\n%d\n%d\n%d",&targ_speed,platform_type,sensor_res,&fov,
&range,&revisit_per,&bars_perframe);
```

Appendix C

```
/* Calculation of derived parameters */

if (strncasecmp(sensor_res,hi,2) == 0)
    one_by_j12 = 1000000/100;
else
    one_by_j12 = 1000000/250;

if (strncasecmp(platform_type,"VF-X") == 0)
    platform_vel = e6;
else if (strncasecmp(platform_type,"VF") == 0)
    platform_vel = f6;
else if (strncasecmp(platform_type,"VP") == 0)
    platform_vel = g6;
else
{
    printf("\n\nSorry - looks like you have a corrupted requirements file. Aborting...\n\n");
    exit(0);
}

clutter_range = 1609.344 * range;

platform_disp = platform_vel * revisit_per;

clutter_motion = 2*(asin(platform_disp/(2*clutter_range)))*one_by_j12;

target_range = 1609.344 * range;

target_vel = d6 * targ_speed;

target_disp = target_vel * revisit_per;

target_motion = 2*asin(target_disp/(2*target_range))*one_by_j12;

targ_movement = target_motion;
clutter_movement = clutter_motion;

return(targ_movement);

}

/*****
/*          Function form_req_file          */
*****/
```

Appendix C

```
form_req_file(fil_nam)
char *fil_nam;

{

FILE *req_file;

float targ_sp;
int foview,rang,revisit,bars;
char *plat_typ,*sen_res;

    plat_typ = (char *) calloc(10,sizeof(char));
    sen_res = (char *) calloc(10,sizeof(char));

req_file = fopen(fil_nam,"w");

printf("\n\nEnter the following independent parameters : \n\n");

printf("Target Speed in Mach\t: ");
scanf("%f",&targ_sp);
printf("\nPlatform Type (VF-X/VF/VP)\t: ");
scanf("%s",plat_typ);
printf("\nSensor Resolution (LOW/HI)\t: ");
scanf("%s",sen_res);
printf("\nFOV in degrees\t: ");
scanf("%d",&foview);
printf("\nRange in miles\t: ");
scanf("%d",&rang);
printf("\nRevisit Period in secs\t: ");
scanf("%d",&revisit);
printf("\nBars per frame\t: ");
scanf("%d",&bars);

fprintf(req_file,"%3.1f\n%s\n%s\n%d\n%d\n%d\n%d",targ_sp,plat_typ,sen_res,foview,rang,revisit,bars);

fclose(req_file);

return;

}
```

Appendix D: Simulation Control Files

Simulation Control File Templates

Simulation Control File for the ON-LINE processing option

```
assign (7) TARG_RANGE
open /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/frame_no.out
open /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/comp.out
open -p Test_Bench_Target
wmove Test_Bench_Target 34,19
wsize Test_Bench_Target 45,40
open -p Test_Bench_O/P
wmove Test_Bench_O/P 34,41
wsize Test_Bench_O/P 45,40
```

```
set FRCNT 1
set FRCNT1 1
set a 0
set b 1
set TEMPX 0
set TEMPY 0
set ZEROS 0
```

```
while ((INIT = '1') and (RUN = '0')) loop
```

```
set a '$a+1'
echo Creating Frame # $FRCNT1 .... > Test_Bench_Target($a,$b)
set a '$a+1'
```

Appendix D

```
set COLS 31
set ROWS 31
set I 0
set J 0
```

```
run $PERIOD
```

```
set FRCNT ` $FRCNT+1`
```

```
while ($I < ` $ROWS+1`) loop
```

```
    while ($J < ` $COLS+1`) loop
```

```
        if (CLUT($I,$J)<= 50) then
            echo 1 > Test_Bench_O/P(` $I+4`, $J)
        end if
        if ((CLUT($I,$J)>50) and (CLUT($I,$J)<= 100)) then
            echo 2 > Test_Bench_O/P(` $I+4`, $J)
        end if
        if ((CLUT($I,$J)>100) and (CLUT($I,$J)<= 150)) then
            echo 3 > Test_Bench_O/P(` $I+4`, $J)
        end if
        if ((CLUT($I,$J)>150) and (CLUT($I,$J)<= 200)) then
            echo 4 > Test_Bench_O/P(` $I+4`, $J)
        end if
        if ((CLUT($I,$J)>200) and (CLUT($I,$J)<= 255)) then
            echo 5 > Test_Bench_O/P(` $I+4`, $J)
        end if
```

```
        set J ` $J+1`
```

```
    end loop
```

```
    set I ` $I+1`
    set J 0
```

```
end loop
```

```
fprint "%d " $ZEROS > /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/comp.out
fprint "%d\n" $ZEROS > /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/comp.out
```

```
echo Frame Number = $FRCNT1 > Test_Bench_Target($a,$b)
set a ` $a+1`
echo No Targets... > Test_Bench_Target($a,$b)
set a ` $a+1`
```

Appendix D

```
fprint "%d\n" $FRCNT1 > /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/frame_no.out
set FRCNT1 `FRCNT1+1`
```

```
end loop
```

```
while (RUN = '1') loop
```

```
set a `a+1`
echo Creating Frame # $FRCNT1 .... > Test_Bench_Target($a,$b)
set a `a+1`
```

```
set COLS 31
set ROWS 31
set I 0
set J 0
```

```
run $PERIOD
```

```
set FRCNT `FRCNT+1`
```

```
while ($I < `ROWS+1`) loop
```

```
while ($J < `COLS+1`) loop
```

```
if (FRAME($I,$J) = 255) then
    set I_TEMP1 `I+4`
    set J_TEMP1 `J`

    echo . > Test_Bench_O/P($TEMPX,$TEMPY)
```

```
set TEMPX `I_TEMP1`
set TEMPY `J_TEMP1`

echo # > Test_Bench_O/P(`I+4`, $J)
```

```
end if
```

```
set J `J+1`
```

```
end loop
```

Appendix D

```
set I '$I+1'
set J 0

end loop

fprintf "%d " X_COORD > /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/comp.out
fprintf "%d\n" Y_COORD > /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/comp.out

echo Frame Number = $FRCNT1 > Test_Bench_Target($a,$b)
set a '$a+1'
echo Target X_coord = 'X_COORD' > Test_Bench_Target($a,$b)
set a '$a+1'
echo Target Y_coord = 'Y_COORD' > Test_Bench_Target($a,$b)
set a '$a+1'

fprintf "%d\n" $FRCNT1 > /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/frame_no.out
set FRCNT1 '$FRCNT1+1'

end loop
set FRCNT1 999
fprintf "%d\n" $FRCNT1 > /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/frame_no.out
set a '$a+2'
echo DONE > Test_Bench_Target($a,$b)

!echo '999' >> /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/images.txt
```

Simulation Control File for the POST processing option

```
assign (7) TARG_RANGE
open /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/frame_no.out
open /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/comp.out

set FRCNT 1
set FRCNT1 1
set ZEROS 0

while ((INIT = '1') and (RUN = '0')) loop

run $PERIOD

set FRCNT '$FRCNT+1'

fprintf "%d " $ZEROS > /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/comp.out
fprintf "%d\n" $ZEROS > /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/comp.out

fprintf "%d\n" $FRCNT1 > /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/frame_no.out
```

Appendix D

```
set FRCNT1 `FRCNT1+1`

end loop

while (RUN = '1') loop

    run $PERIOD

    fprintf "%d " X_COORD > /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/comp.out
    fprintf "%d\n" Y_COORD > /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/comp.out

    fprintf "%d\n" $FRCNT1 > /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/frame_no.out
    set FRCNT1 `FRCNT1+1`

end loop

set FRCNT1 999

--fprintf "%d\n" $FRCNT1 > /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/images.txt
fprintf "%d\n" $FRCNT1 > /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/frame_no.out

!echo '999' >> /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/images.txt

quit
```

A Final Simulation Control File created by the IUI using the post processing template

```
cd TB22
assign 12 X_COORD
assign 12 Y_COORD
assign 2 VELOCITY
assign 3 MODE_MOTION
assign 45 ANGLE
assign ('1') INIT
assign -after 2 ('1') RUN
assign -after 10 ('0') RUN
assign -after 10 ('0') INIT
assign (2 ns) PER
set PERIOD 2
assign (7) TARG_RANGE
open /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/frame_no.out
open /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/comp.out

set FRCNT 1
set FRCNT1 1
set ZEROS 0

while ((INIT = '1') and (RUN = '0')) loop
```

Appendix D

```
run $PERIOD

set FRCNT `FRCNT+1`

fprintf "%d " $ZEROS > /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/comp.out
fprintf "%d\n" $ZEROS > /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/comp.out

fprintf "%d\n" $FRCNT1 > /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/frame_no.out
set FRCNT1 `FRCNT1+1`

end loop

while (RUN = '1') loop

run $PERIOD

fprintf "%d " X_COORD > /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/comp.out
fprintf "%d\n" Y_COORD > /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/comp.out

fprintf "%d\n" $FRCNT1 > /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/frame_no.out
set FRCNT1 `FRCNT1+1`

end loop

set FRCNT1 999

--fprintf "%d\n" $FRCNT1 > /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/images.txt
fprintf "%d\n" $FRCNT1 > /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/frame_no.out

!echo '999' >> /home/RASSP/DEMO_THESIS/2D_IMAGE_PROC/images.txt

quit
```

Appendix E: Code Listing for C program to Measure CPU Time

("time1.c")

```
#include <stdio.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <sys/types.h>

#ifdef __hpux
#include <sys/syscall.h>
#define getrusage(a, b) syscall(SYS_GETRUSAGE, a, b)
#endif /* hpux */

void PrintTime(fp, ptv)
FILE *fp;
struct timeval *ptv;
{
    if (ptv->tv_sec >= 2000) {
        fprintf(fp, "%u ms", (ptv->tv_usec+500) / 1000 + ptv->tv_sec * 1000);
    } else {
        fprintf(fp, "%u us", (ptv->tv_sec * 1000000) + ptv->tv_usec);
    }
}

main(argc, argv)
int argc;
char *argv[];
```

Appendix E

```
{
    int rc;
    struct rusage ru;
    struct timeval tvStart, tvStop;

    if (argc < 2) {
        fprintf(stderr, "need something to run!\n");
        return 1;
    }

    gettimeofday(&tvStart, 0);
    switch(fork()) {
        case -1:
            perror("fork failed");
            exit(1);
        case 0:
            execvp(argv[1], &argv[1]);
        default:
            wait(0);
    }
    gettimeofday(&tvStop, 0);
    memset(&ru, 0, sizeof(ru));
    getrusage(RUSAGE_CHILDREN, &ru);

    fprintf(stderr, "\treal time ");
    tvStop.tv_sec -= tvStart.tv_sec;
    tvStop.tv_usec -= tvStart.tv_usec;
    if (tvStop.tv_usec < 0) {
        tvStop.tv_sec--;
        tvStop.tv_usec += 1000000;
    }
    PrintTime(stderr, &tvStop);

    fprintf(stderr, "\n");
    fprintf(stderr, "\tuser time ");
    PrintTime(stderr, &ru.ru_utime);
    fprintf(stderr, "\n");
    fprintf(stderr, "\tsyst time");
    PrintTime(stderr, &ru.ru_stime);
    fprintf(stderr, "\n");

    if (!strcmp(argv[0], "rttime")) {
        /*fprintf(stderr,
            "\tmaxrss %d\n"
            "\tintegral shared memory size %d\n"
            "\tintegral unshared data size %d\n"
            "\tintegral unshared stack size %d\n"
            "\tpage reclaims %d\n"
            "\tpage faults %d\n"
            "\tswaps %d\n"
            "\tblock input operations %d\n"

```

Appendix E

```
        "\tblock output operations %d\n"  
        "\t# of characters read/written %d\n"  
        "\tmessages sent %d\n"  
        "\tmessages received %d\n"  
        "\tsignals received %d\n"  
        "\tvvoluntary context switches %d\n"  
        "\tinvoluntary context switches %d\n",  
        ru.ru_maxrss,  
        ru.ru_ixrss,  
        ru.ru_idrss,  
        ru.ru_isrss,  
        ru.ru_minflt,  
        ru.ru_majflt,  
        ru.ru_nswap,  
        ru.ru_inblock,  
        ru.ru_oublock,  
        ru.ru_ioch,  
        ru.ru_msgsnd,  
        ru.ru_msgrcv,  
        ru.ru_nsignals,  
        ru.ru_nvcsw,  
        ru.ru_nivcsw)*;/  
    }  
    return 0;  
}
```

Vita

Prabhakar Gowrisankaran was born in Bombay, India on August 10, 1972. He graduated with a Bachelor of Engineering degree in Instrumentation from the Birla Institute of Technology and Science, Pilani, India in June 1993. He attended graduate school at Virginia Polytechnic Institute and State University and received a Master of Science degree in Electrical Engineering in March 1995. He has been employed with COMSAT Labs, Maryland since March 1995.

A handwritten signature in black ink, appearing to read 'Prabhakar', with a long horizontal stroke underneath.