A Test Plan Driven Test Bench Generation System

by

Sailesh Kottapalli

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. W. R. Cyre

May 1996

Blacksburg, Virginia
A Test Plan Driven Test Bench Generation System

by

Sailesh Kottapalli

Dr. James R. Armstrong, Chairman

Electrical Engineering

(ABSTRACT)

Testing and verification of large DSP models is a laborious and time consuming task. Test benches provide a platform for testing VHDL models. Development of good test benches is very critical in reducing the time, manpower and costs involved in testing of such models. Sometimes the development of test benches alone does not make the task of testing easy. High level approaches have to be developed to configure the test bench according to the required testing strategy. This would relieve the user of familiarizing himself with the details of the models to configure the test bench properly for a particular testing scenario.

A test plan organizes the system requirements in terms of how these requirements will be tested. It divides the system requirements into groups and allocates a set of tests to each of the requirements groups. The main emphasis of this thesis is to develop an approach to interface a test bench with a test plan, which configures the test bench according to the test to be performed.

As an illustration of this approach a test plan interface was developed for a test bench system for an Infrared Search and Track (IRST) algorithm. A requirements interface was designed to convert the primary parameters to the form required by the test
bench primitives. Enhancements were made to the previous version of the test bench primitives and a structural test bench was developed by the interconnection of the test bench primitives using a commercial schematic capture tool, Synopsys Graphical Environment (SGE). A test plan interface was designed to configure the test bench with the required input parameters for creation of test vectors according to the test plan. The test bench can be operated in iterative mode to find the limiting value of a system parameter.

This thesis also describes the integration of the different elements of the test bench, developed using different computer tools, and programming languages into a single test bench generation system. A menu driven X-Windows user interface was also developed to facilitate easy operation of the test bench generation system.
To my parents
Acknowledgments

I wish to express my whole hearted thanks to Dr. James R. Armstrong for his guidance, encouragement and support. It has been an honor to study and work under him. I would like to express my gratitude to Dr. F. Gail Gray for all his help and for serving as a member on my committee. I would also like to thank Dr. Walling R. Cyre for being on my graduate committee.

Thanks also go to all the people working in the VISC lab for making it an enjoyable workplace. Special thanks to John and Morris for all their help. I would like thank my parents and my brother for all the moral support and for having stood by me at every step in my life.
Table of Contents

Chapter 1. Introduction ...........................................................................................................1
  1.1 Motivation .....................................................................................................................1
  1.2 RASSP Test Bench Generation Task ..........................................................................3
  1.3 Contributions .............................................................................................................4
  1.4 Thesis Organization ......................................................................................................7

Chapter 2. Background .........................................................................................................9
  2.1 Overview On Test Benches .........................................................................................9
  2.2 Test Vector Generation Techniques ........................................................................12
  2.3 IRST systems .............................................................................................................13
     2.3.1 IRST Stimulus Generator ....................................................................................14
  2.4 Previous Work ..........................................................................................................16

Chapter 3. Requirements Interface ..................................................................................20
  3.1 Introduction ..............................................................................................................20
  3.2 Specification Repository ...........................................................................................21
  3.3 Primary And Secondary Requirements For IRST System .......................................23
  3.4 Math Model For Requirements Interface ..................................................................31

Chapter 4. IRST Test Bench Primitives and Model Development ...................................35
  4.1 Introduction ..............................................................................................................35
List of Illustrations

Figure 2.1. A Test Bench ........................................................................................................9
Figure 2.2. An off-line test bench configuration .................................................................10
Figure 2.3. On-line, non-adaptive test bench configuration .............................................11
Figure 2.4. Block Diagram of a typical IRST system .......................................................13
Figure 2.5. Physical structure of an IRST sensor ............................................................15
Figure 2.6. Simulation model of IRST stimulus generator ................................................16
Figure 3.1. A high level test bench generation system ......................................................21
Figure 3.2. System level schematic of IRST system showing specification modification ..............................................................................................................23
Figure 3.3. Requirements interface for IRS ....................................................................24
Figure 3.4. Illustration of platform direction ..................................................................25
Figure 3.5. Overview of target initial position ................................................................27
Figure 3.6. Three dimensional coordinate geometry used to represent target trajectory ..28
Figure 3.7. Sensor Gain Variation ....................................................................................29
Figure 4.1. VHDL test bench for IRST algorithm ..............................................................35
Figure 4.2. Illustration of stimulus generator for IRST algorithm ..................................37
Figure 4.3. The Environment Component .......................................................................39
Figure 4.4. The Target Component .................................................................................41
Figure 4.5. The Sensor Component ................................................................................42
Figure 4.6. Illustration of IRST algorithm .......................................................................47
Figure 5.1. A high level test bench generation system with test plan interface ..........52
Figure 6.1. Test bench generation system .......................................................60
Figure 6.2. IRST test bench system user interface menu structure ......................62
Figure 7.1. Test vectors used in the file I/O test case ........................................66
Figure 7.2. MUT outputs for the file I/O test case ............................................67
Figure 7.3. Comparator outputs for the file I/O test case ....................................68
Figure 7.4. Clutter selected for code-generated test case ....................................69
Figure 7.5. Test vector generated in the code-generated test case .......................70
Figure 7.6. MUT outputs for the code-generated test case .................................71
Figure 7.7. Comparator outputs for the code-generated test case .......................72
List of Tables

Table 5.1. Requirements Grouping for Test Group 1 ........................................53
Table 5.2. Requirements Grouping for Test Group 2 ........................................55
Table 5.3. Requirements Grouping for Test Group 3 ........................................56
Table 5.4. Parameters on which iterative test is performed .................................58
Table 7.1. Limiting values of parameters for IRST MUT ...................................72
Chapter 1. Introduction

1.1 Motivation

In the last decade the field of digital design and VLSI design has undergone a rapid change. The complexity of the circuits being built in a single chip has been increasing exponentially over this period to such an extent that now the number of transistors on a single chip has reached the millions. This has made the design of a chip at the transistor level virtually impossible [1]. The duration of the product development cycle, from concept forming, to fabricating the chip on silicon is to be kept short to maintain competitiveness [18]. As a result, designers have had to follow new approaches to design hardware. The most widely used approach is the creation of simulation models of the system with the help of computer tools. A critical factor in the creation of these simulation models is the capability to accurately describe the behavior of the design. Software languages like C and PASCAL are inadequate for this purpose, since they do not have the ability to model concurrency in the execution of logic blocks. Furthermore these languages cannot represent the actual timing relationships between different logic elements. Out of these requirements arose several specialized hardware description languages such as
AHPL [31] and ISP [32]. These specialized languages evolved into the VHISC Hardware Description Language (VHDL), which has been endorsed by the IEEE as a standard [3].

Hardware description languages allow the user to create computer models of hardware at different levels of the design hierarchy [2]. These languages can model concurrency in the execution of the logic blocks and also model the timing relationship between the different logic constructs. Hardware description languages provide a convenient tool for the logic specification of the digital circuits, reducing the amount of information to be handled by the designer and thereby reducing the time taken to design hardware. Once the model is constructed, it can be simulated with a set of test vectors to verify the functionality of the model.

The VHISC Hardware description language (VHDL) is very popular in the electronic industry as well as research labs because of the rich constructs that it provides for the accurate modeling of the digital circuits either in top-down or bottom-up design at varying levels of abstraction. VHDL supports both behavioral and structural modeling.

Design verification of these models forms a very important part of the life cycle of a system design. With the rapid increase in the amount of functionality being incorporated into a single chip the amount of time taken to test and verify the design has also increased. The important factor in testing different operations of the model is the generation of appropriate test stimuli. The general practice in the verification process is to simulate the model with a large set of test data manually developed either by the designer or the model developer. The generation of this test data is a very time consuming and a labor intensive task. Also, the test data so developed satisfies no formal definition of completeness [4]. In relieving the user of such problems, use of high level test generation techniques have been proposed [5-12] for VHDL models.
A major focus of the RASSP program here at Virginia Tech is the development of VHDL models for signal processing algorithms. As mentioned earlier, the models 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 complex task. Therefore there is a critical need to develop high level approaches for development of the VHDL test bench to relieve the modeler of the details of this time consuming task. Also, the model tests created by the user using the VHDL test bench may not accurately test all the operations of the model. Thus it is also very important to develop high level approaches to configure the test bench automatically according to the test to be performed. The test bench would then generate the required test vectors and execute the model with it. This would relieve the user of the detailed understanding of the working of the test bench to create test stimuli for any particular test case.

1.2 RASSP Test bench Generation task

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 [13].

VHDL modeling is an important part of this process. Testing these models involves the use of a test bench, which is defined as follows:

An executable VHDL model that instantiates a model under test (MUT) and provides the capability to drive the MUT with a set of test vectors. A test bench is the top level in the model hierarchy and is the entity simulated when testing the model [14].
The basic approach to test bench development developed at Virginia Tech as a part of the RASSP program is as follows [14, 15, 16]:

1. Test bench code elements are developed using high level design tools.

2. Environmental data generators are used to prepare the input files that can be read by the test bench. These environmental data generators can produce very realistic inputs to the Model Under Test (MUT).

3. The system specification is linked to the test bench through a requirements interface which allows the use of a test plan.

4. A test plan interface configures the test bench to match the requirements for a particular test in the test plan.

5. The comparator compares the simulation results with the expected response from the test bench to check if the response of the MUT is correct or not.

6. All these modules are joined together by an integration program to form a test bench system.

1.3 Contributions

The primary objective of this thesis is to develop a methodology to interface a test bench with a test plan, wherein a user can specify a particular situation in which a model has to operate or a particular aspect of the model to be determined, e.g., what is the maximum noise resolution of the MUT. The test plan configures the test bench to generate a test
vector or a series of test vectors which facilitate the evaluation of the particular aspect of the MUT.

As an example of this approach, the test plan interface was designed for a test bench for an Infrared Search and Track (IRST) algorithm. A primitive IRST test bench and IRST model were developed here at Virginia Tech in the previous version of Test Bench Generator (TBG, version 1) [17, 18 19]. The following enhancements were made to the IRST test bench and the IRST model in version 2 of TBG as a part of this work:

1. A specification repository was designed to facilitate easy specification of the system parameters by the system engineer. A C program was developed to extract the system parameters automatically from the specification repository.

2. A requirements interface was designed to convert the system parameters to the form required by the test bench.

3. Platform motion and sensor scan movement were incorporated into the test bench, i.e., the test bench was provided with the capability to produce a sequence of sensor frames with platform motion and sensor scan motion.

4. An error module was developed to incorporate noise or sensor bias into the sensor frames.

5. IRTool [26] is used to generate two dimensional, infrared images of targets as detected by infrared sensors. A C program was developed to generate an IRTool execution script, according to the target specifications. The IRTool execution script controls the execution of the IRTool software.
6. IRAMMP [20], a data base of infrared clutter files, i.e., files of two dimensional images which portray sea and sky clutter, maintained for the Naval Research Lab, is used to serve as a repository of different background clutter.

7. The IRST algorithm [19] was modified to identify sensor frame movement in the successive input frames (produced as a result of platform motion as well as sensor horizontal scan). The target motion of both single point as well as multiple point target signatures can be detected along with the probable target signature.

8. A comparator was developed to compare the simulation results of the model under test with the expected response from the test bench, in order to determine the correctness of the simulation results.

9. A test plan interface was designed to configure the test bench to match the particular test in the test plan to be performed.

10. A C program was developed to generate a VHDL simulation control file, used to control the simulation of the VHDL model according to the test plan configuration.

11. An X-Windows interface was designed to provide a user friendly interface to the test bench system as well as to integrate all the above mentioned pieces of the test bench into a single IRST test bench system. The above modules were coded in different environments. The test bench and the MUT are completely in VHDL, most of the interface modules were developed in C language, all the display programs were coded in Matlab [29] and some part of the system was coded using C Shell script. The X-Windows interface serves the purpose of integrating these different modules under different platforms into a single system. This system has been developed for a UNIX platform.
1.4 Thesis Organization

This thesis consists of an abstract, 8 chapters, bibliography and appendices.

Chapter 1, “Introduction,” provides the motivation for the thesis, an overview of the IRST test bench generation task, and a list of the main contributions of the thesis.

Chapter 2, “Background,” provides an overview of the test benches and Infrared Search and Track systems. It discusses the test bench generation requirements, test vector generation techniques, work done previously in this area, and its limitations.

Chapter 3, “Requirements Interface,” discusses the development of a requirements interface and a specifications repository to store the primary system requirements.

Chapter 4, “IRST Test Bench Primitives And Model Development,” outlines all the modifications and enhancements that the writer has made to the previous version of the IRST test bench primitives and the IRST model.

Chapter 5, “Test Plan Interface And Iterative Testing”, discusses how the test bench is configured by the test plan interface to match a particular test in the test plan, and automatic generation of test vector or a series of test vectors using the iterative mode of testing.

Chapter 6, “System Integration And User Interface Development," describes integration of all the different pieces of the test bench and development of an X-Windows program, which performs the task of providing the user-interface to the test bench.

Chapter 1. Introduction
Chapter 7, “Results,” provides the simulation results of the IRST test bench and gives a brief description of the simulation results.

Chapter 8, “Conclusions And Future Work,” draws conclusions regarding the methodology and implementation of test bench system and suggests areas of future work.
Chapter 2. Background

2.1 Overview On Test Benches

As stated in section 1.1, a test bench is an executable VHDL model that instantiates the model being tested, provides the capability to drive the MUT with a set of test vectors and compares its response with the expected response. Test benches thus provide the user with a capability to test the MUT thoroughly by simulation [14]. Figure 2.1 shows a general test bench. The stimulus generator generates the test vectors and drives the model under test (MUT) with the test vectors, the model response and the expected result from the stimulus generator are sent to a comparator and appropriate Go/No Go signal is generated.

![Figure 2.1 A Test Bench](image)

Chapter 2. Background
There can be two types of feedback. First, the model under test can feed its state back to the stimulus generator, allowing interaction between the two elements. Second, the comparator signals can be fed back to the stimulus generator to allow adaptive testing. This second type of feedback has been implemented in this thesis. A test bench generation process should support at least three basic test bench configurations: off-line, on-line and adaptive. The configurations are explained below.

1. Off-line test benches [14]
In an off-line test bench, the stimulus generator reads a file of test stimuli, and at the appropriate time sends the test vectors through the interface to the signal processor [14]. Figure 2.2 shows an off-line test bench configuration. In an off line test bench the comparator reads the expected response and alerts the user if the actual response is different from the expected response.

![Figure 2.2 An off-line test bench configuration](image-url)
The input files may be real data from the system being modeled or data files developed using a commercial modeling tool.

2. On-line, non-adaptive test benches [14]
In this configuration the stimulus generator generates the test vectors which are fed to the MUT on-line, i.e., as the test vectors are created. The stimulus generator feeds the test vectors to a gold model, i.e., a model that is assumed to produce the correct response to the stimuli, of the MUT also. The response of the MUT to test vectors are compared with the expected response from the gold model, on-line. The comparator produces the appropriate Go/No Go signal. The test vectors generated along with the expected response can be stored to drive the MUT later in off-line configuration. This configuration is non-adaptive because the stimulus generator does not react to the response of the MUT and/or comparator. Figure 2.3 shows an on-line, non-adaptive test bench configuration.

![Figure 2.3 On-line, non-adaptive test bench configuration](image)
3. Adaptive test benches [14]

Adaptive test benches are the most complex type of test benches. An adaptive test bench reacts to the feedback from the comparator and/or the MUT as it computes the stimuli during the simulation, i.e., the stimulus generator modifies the test vectors generated based on the response of the MUT and/or the comparator. Figure 2.1 illustrates an adaptive test bench.

The test benches developed as a part of this thesis can be operated in all three configurations listed above. The actual implementation of these configurations is described in detail in Chapter 6.

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 appropriately or not [21]. Test generation is defined as the process of determining the test stimuli necessary to test a system [21].

Two basic approaches to test vector generation are:

1. Model based approach: In this approach the VHDL MUT itself is used to produce the tests. Two types of tests generated with this approach are - model perturbation with high level faults [22] and I/O path sensitization [23].
2. Environment based approach: This approach uses the environment surrounding the model to generate the test vectors. The environment is modeled as a part of the test bench to generate the normal system inputs.

The test bench developed as a part of this thesis uses the environment based test generation because the type of inputs experienced by the signal processing models are a complicated function of the model environment [24] and could not be generated from the model itself.

2.3 IRST Systems

Infrared Search and Track (IRST) systems are a functionally defined subset or class of passive military infrared sensor systems whose objective is to reliably detect, locate, and continuously track infrared-emitting objects and targets in the presence of background radiation (clutter) and other disturbances [25].

![Figure 2.4 Block diagram of a typical IRST system [25]](image-url)
An IRST system consists of an infrared scanning sensor, IRST computer with real-
time signal processing electronics and optical band-pass filters, a display and other output
functions [25]. Figure 2.4 shows a block diagram of a basic IRST system.

The primary objective of the IRST algorithm is to detect moving targets in a sequence of
infrared image frames of any arbitrary size, that 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 successive image frame in all directions within a
fixed scan range [18]. Based on the best alignment, the speed of the target, the movement
of the sensor frame, caused by either the scanning action of the sensor or by the motion of
the platform or a combination of both, and the probable target signatures are determined.
Thus, the task of a stimulus generator for an IRST algorithm is to provide the algorithm
with a sequence of infrared image frames, with target signature information merged in a
background clutter. Figure 2.5 illustrates the physical structure of an IRST sensor. An
infrared sensor is mounted on a platform. The platform can be a ship, airplane or a tank.
The sensor can scan horizontally about its axis. The sensor captures the infra-red image of
the background clutter and target within its field of view as it scans and/or the platform
moves. The captured images (termed as sensor frames) are sent to the IRST algorithm for
detection and tracking of the possible target.

2.3.1 IRST Stimulus Generator

The stimulus generator is the main component of the IRST test bench. The
stimulus generator models the environment surrounding the IRST algorithm and generates
a sequence of image frames that contain both the target and the background clutter. The
physical structure modeled by the stimulus generator is shown in figure 2.5. A computer
simulation model of the stimulus generator is to be developed to emulate the performance
of the real infrared sensor.
Figure 2.5 Physical structure of an IRST sensor

Figure 2.6 shows the structure of the typical simulation model of the IRST stimulus generator. The main sub-components of the stimulus generator are: an oscillator, a target component, an environment component, and a sensor component. A brief description of the function of each sub-component is provided below.

Oscillator component: This component synchronizes the actions of all the other sub-components of the stimulus generator. It also controls the rate of generation of the image frames.

Environment component: This component performs the task of development of the background clutter. The clutter information could be infrared (IR) images of sea, sky and land. The environment component can also take information from the sensor about the location and the orientation of the platform and the sensor to generate the clutter information.

Chapter 2. Background
Figure 2.6 Simulation model of IRST stimulus generator

**Target component:** This component performs the task of generating an infrared signature of the target, models the trajectory of the target, and also merges the target information with the background clutter generated by the environment component.

**Sensor component:** This component models the sensor scan and platform motion, captures the corresponding frames from the merged background clutter and injects noise into the frames.

The actual implementation of the VHDL model of each sub-component and how they work is explained in detail in Chapter 4.

**2.4 Previous Work**

A software tool, Test Bench Generator (TBG, Version 1) was developed here at Virginia Tech as a part of the research work done on the RASSP program. The tool was used to create the VHDL test bench and the test vectors for Infrared Search and Track
IRST) systems. The work of Hrishikes [7], Prabhakaran [18], and Commissariat [19] describes the creation and the use of the IRST test bench and the IRST model. There were several limitations and oversights in this version of the IRST test bench. Among the limitations are:

1. The target component produces only a single point target and is always represented by a pixel of intensity level 255 on the gray scale. In a real scenario the targets are mostly multi-point, i.e., target signature is represented by more than one pixel. Also, the intensity of the pixels representing the target is dependent on the temperature of that target. The higher the temperature, the higher is the intensity of the pixels on the gray scale (0-255). As the temperature of the target is not constant throughout its surface, the target signature must be represented by multiple pixels with varying intensity levels.

2. The target module can model a very restricted trajectory of target motion. Also, The target is always placed at the bottom left corner of the first frame of the sequence. There were also some bugs in the calculation of the target displacement that caused the wrong target displacement in successive frames in some test cases.

3. The environmental module is responsible for creating the background clutter, or loading the background clutter from the file in the database. The environmental module produces a clutter of size 32x32, whereas in a working system it would be more appropriate to have a clutter of variable size. Also, no realistic clutter information can be depicted in an image frame of size 32x32. It is necessary not only to be able to generate larger image frames, but also be able to vary the size of the image frames based on parameters like sensor field of view and/or clutter range. Furthermore, the image frames generated are not infrared clutter images as would be encountered in an IRST system.
4. The sensor module takes as input the clutter information (2 dimensional array of size 32x32), the target information (2 dimensional array of size 32x32, containing target pixel at the position defined by the actual position of the target with all the other array values kept at zero) and merges both of them to form the sensor image frame. The function of most real sensors is to capture the image of the merged clutter, i.e., clutter with target merged in it, within its field of view. Therefore a more appropriate modeling approach would be to present the sensor module with the background clutter with target information already merged in it. The sensor module would then capture a part of this image as a sensor frame, rather than receiving both target information and clutter information separately and merge both these together itself. The size of the sensor frame captured from merged background clutter would be dependent on the field of view of the sensor and the clutter range.

5. All sensors add noise to the captured image frame. The noise could be either due to some mechanical action, e.g., yaw, roll, pitch, or due to the electronic processing involved while capturing of the frame, e.g., quantization, noise, etc. Therefore it is necessary for the test bench to have the capability to add noise. The previous version of the test bench did not support this.

6. The IRST test bench can model only standing platforms and staring sensors.

7. The test bench generates the test stimuli and the expected response. The MUT processes the stimulus and outputs its response. Once the MUT simulation results are obtained the test bench does not have the ability to compare the MUT results with the expected response to verify the operation of the MUT. This can be achieved by developing a comparator model, which would automatically compare both the responses. The previous version of the test bench did not have this feature.
8. The IRST model performs the task of identifying and tracking the target in the continuous sequence of frames. The previous version of IRST model can detect only single point targets. In a real scenario we would like the model to be able to detect multi-point target signature. Also, the metric used to find the sensor frame shift and the target motion does not perform consistently in all circumstances, especially with multi-point target signatures.
Chapter 3. Requirements Interface

3.1 Introduction

An important consideration in the development of test benches is that the test bench has to be an immediate interpretation of the system specifications. The system specifications consist of general requirements and specific requirements which together specify the system being modeled.

General requirements specify the class of system being modeled, for example, these requirements can be used to select appropriate target, environment and sensor models from the VHDL test bench library. General requirements may also be used to specify the behavior of new components that must be input to high level design tools and code generators, which are used to generate the VHDL code for the test bench [4]. Specific requirements select a particular member of the class of systems, i.e., they define the key parameters of the components that form the class of system specified by general requirements [4]. In general, these specific requirements are not expressed in the form required by the test bench. The specific requirements as expressed in the specification are considered to be primary. The requirements interface performs the task of converting these primary requirements to secondary requirements, which are in the form needed by the test bench. Both the primary and the secondary specific requirements become values...
for test bench generics. Figure 3.1 depicts how the general and the specific requirements fit into a high level test bench generation system.

![Diagram of high level test bench generation system]

**Figure 3.1 A high level test bench generation system**

The requirements interface uses a math model specific to the system being modeled to derive the secondary requirements from the primary requirements. This approach has been adopted to form a requirements interface for the IRST test bench development.

### 3.2 Specification Repository

An important consideration is how the specific requirements will be stored. The personnel dealing directly with the specific requirements are frequently not knowledgeable in the VHDL language [27]. Traditionally these specifications are stored in a specifications document. The system engineer retrieves these specifications from the
documents and feeds them to the system manually. This kind of requirements specification is error prone and time consuming. Therefore, a specification repository was developed to relieve the system engineer of this manual task. The development of the specification repository is as follows:

A top level schematic diagram of the IRST test bench system is developed as an interconnection of basic blocks corresponding to real system components. Each block represents a major component of the real system and has a set of system parameters associated with it. The user can check the actual values of the system parameters and change them as applicable. This kind of facility to specify the system parameters would provide the system engineer with a better perspective of the system parameters. Also, this kind of representation facilitates automatic retrieval of the specifications.

The schematic diagram was developed using a commercial schematic capture tool, Synopsys SGE (Synopsys Graphical Environment) [28]. SGE provides a VHDL interface to generate the VHDL description of the design created using schematic capture. All the individual blocks in the schematic are configured as VHDL entity/architecture constructs, and each of the system parameters of the blocks is described as a VHDL generic. A parser was developed in C to automatically parse the VHDL code dumped by SGE and extract the names and the corresponding values of the system parameters. These parameters are input to the requirements interface. The high level schematic diagram created using SGE is depicted in figure 3.2. The figure also shows how the specific parameters corresponding to each block are represented.
Figure 3.2 System level schematic of the IRST system showing specification modification.

3.3 Primary And Secondary Requirements For IRST System

Figure 3.3 depicts the requirements interface for IRST. The primary requirements for the IRST application are:
**Frame Rate:** This parameter defines the rate at which the frames are to be generated. The parameter is specified in no of frames per second.

**Number of frames:** The number of frames to be generated in the particular test case.

![Diagram of Requirements Interface for IRST](image)

**Figure 3.3 Requirements interface for IRST.**

**Sensor Field of View:** The angle within which the sensor captures the frame. This parameter is specified in milliradians.

**Clutter Range:** The distance of the clutter in kilometers from the platform.
Platform Type: This defines whether the platform is a standing platform or a moving platform.

Platform Speed: The speed of the platform in meters per second.

Platform Direction: This parameter defines the direction of the platform in a 2 dimensional plane formed by the horizontal and vertical axes as is depicted in the figure 3.4. The direction is specified as the angle made by the platform trajectory with the Horizontal axis (angle $\theta$ in figure 3.4).

Figure 3.4 Illustration of platform direction.

Sensor Type: This defines whether the sensor is a staring sensor or a scanning sensor. Only horizontal sensor scan has been incorporated. The sensor can scan in both clockwise and counter-clockwise direction.
**Sensor Scan Rate:** This parameter defines the angular velocity at which the sensor is scanning. This parameter is specified in milliradians per second. A positive value indicates a clockwise scan and a negative value indicates a counter-clockwise scan.

**Target Type:** This defines whether the target signature is a single point signature or a signature generated by the IRTool. IRTool usually generates a multi-point target signature.

**Target Range:** This parameter defines the distance of the target from the platform in kilometers.

**Target Speed:** The speed of the target in meters per second.

**Target Position:** This defines the initial position of the target with respect to the sensor location. Two parameters Target_ang_Hz and Target_ang_Vt are used to define the angle subtended by the initial target position with the platform location. Figure 3.5 illustrates how these two parameters define the initial target position. The parameter Target_ang_Hz is the angle made by the line joining the platform’s initial position and the projection of the target’s initial position on the horizontal axis, with the range axis. The parameter Target_ang_Vt is the angle made by the line joining the platform’s initial position and the projection of the target’s initial position on the vertical axis, with the range axis. Both these angles are measured in milliradians.
Figure 3.5 Overview of target initial position

_Target Direction:_ This defines the trajectory of the target in a three dimensional space formed by the horizontal, vertical and range axes. The three dimensional coordinate geometry used to define the target trajectory is illustrated in figure 3.6. Two parameters Target_dir_HzVt and Target_dir_Rn are used to define the trajectory. The parameter Target_dir_HzVt is the angle made by the projection of the target trajectory on the plane formed by the horizontal and vertical axes, with the horizontal axis (angle $\alpha$ in figure 3.6). The parameter Target_dir_Rn is the angle made by the target trajectory with the range axis (angle $\beta$ in figure 3.6).
Figure 3.6 Three dimensional coordinate geometry used to represent target trajectory

*Error Type, Error Level:* These parameters define the kind of errors that the sensor induces in the frames captured from the merged background clutter (clutter information with target information merged in it). Three different types of errors are possible. These errors are as explained below:

(i) **Fixed offset:** In this kind of error the gray scale intensity level of each pixel in the sensor frame is varied by the value specified by the error level. The error level is an integer representing the value of the offset. If the error level is positive then the intensity of each pixel is increased by that amount, and if the error level is negative then the intensity of each pixel is decreased by that amount. In any case the maximum gray scale intensity is 255 and the minimum is 0, thus the final intensity of any pixel has to be within this range.
(ii) Relative offset: In this type of error the gray scale intensity level of each pixel is varied relative to the actual value by an amount specified by the offset level, i.e., the offset level is specified as a percentage.

(iii) Noise: In this type of error the gray scale intensity of some randomly selected pixels are varied by the amount specified by the noise level. The number of pixels selected is also random.

Sensor Gain Variation: A function has been designed to vary the gain of the sensor across the sensor frame, i.e., vary the sensor frame pixel intensities with uneven offsets. The offset introduced by this function is illustrated in figure 3.7. The actual value of the offset is defined by the parameter, sensor gain variation (Y in figure 3.7). The offset is minimum at the center and maximum at the edges of the frame. If the value of the parameter sensor gain variation is 1 then the offsets introduced are 10, 20, 30 and 40 as shown in the figure. The significance of this function is explained in chapter 4.

![Sensor Frame Diagram]

Figure 3.7 Sensor gain variation.
The derived secondary parameters are:

*Frame Time*: This is the time available to generate each sensor frame in milliseconds.

*Simulation End Time*: Time after the start of the simulation that the test bench should stop simulating. This is represented in milliseconds.

*Target Initial Position*: The initial position of the target signature in terms of the pixel position in the 2 dimensional array representing the background clutter. This is represented by two parameters Target_pos_rows and Target_pos_columns, corresponding the two subscripts of the two dimensional array of background clutter.

*Target Displacement*: The displacement of target from one frame to another frame in terms of number of rows and columns in the 2 dimensional array representing the background clutter. This is represented by two parameters TargetDisp_Hz and TargetDisp_Vt, corresponding to the number of columns moved in the horizontal direction and the number of rows moved in the vertical direction respectively.

*Sensor Frame Initial Position*: This is the array subscripts of the 2 dimensional array that represent the top left corner of the sensor frame. This is represented by two parameters SensorFr_pos_rows and SensorFr_pos_columns, corresponding to the column number and row number of the 2 dimensional array, whose intersection forms the top left corner of the sensor frame.

*Sensor Frame Displacement*: The displacement of sensor frame in the background clutter in successive frames of the test vector. This is represented by two parameters SensorFrDisp_Hz and SensorFrDisp_Vt, corresponding to the number of columns and
rows moved respectively. The sensor frame motion is a result of the combined action of platform motion and sensor scan motion.

Sensor Frame Size: This is the size of the sensor frame in terms of number of rows and columns of the 2 dimensional array representing the background clutter. Two parameters are used to represent this. SensorFr_no_rows and SensorFr_no_columns represent the number of columns and number of rows in the sensor frame respectively.

Clutter Size: The requirements interface reads the clutter file to determine the size of the clutter in terms of number of rows and columns representing the 2 dimensional array of the clutter. The actual file read by the requirements interface depends on the clutter selected by the user from the database of clutter files. Clutter size is represented by two parameters Clutter_size_rows and Clutter_size_columns, corresponding to the number of row and number of columns in the 2 dimensional image of the clutter respectively.

Target Size: The requirements interface reads the target signature file to determine the size of the target in terms of number of rows and columns. This is represented by two parameters Target_size_rows and Target_size_columns, corresponding to the number of rows and number of columns in the 2 dimensional image of the target respectively.

3.4 Math Model For Requirements Interface

The math model is a collection of equations and formulas used to derive the secondary parameters from the primary parameters. The math model was first modeled in Matlab and then coded in C. Shown below are the equations used to derive the secondary parameters.

Frame Time = (1000/Frame Rate) milliseconds.
Simulation End Time = (Frame Time x Number of Frames) milliseconds

Target Initial Position

\[
\text{temp} = \frac{(\text{Target Range} \times 1000)}{\text{Sensor Resolution}}
\]

(i) \[\text{Target_pos_columns} = \text{integer} \left( \frac{\text{Clutter_size_columns}}{2} + \sin(-\text{Target_ang_Vt}) \times \text{temp} \right)\]

(ii) \[\text{Target_pos_rows} = \text{integer} \left( \frac{\text{Clutter_size_rows}}{2} + \sin(-\text{Target_ang_Hz}) \times \text{temp} \right)\]

Target_pos_rows and Target_pos_columns are integers because they basically represent the subscripts of a 2 dimensional array.

Target Displacement

\[
\text{temp} = \frac{(\text{Target_dir_Rn} \times \pi)}{180}
\]

\[
\text{temp1} = \text{Target Speed} \times \sin(\text{temp})
\]

\[
\text{temp2} = \frac{(\text{Target_dir_HzVt} \times \pi)}{180}
\]

\[
\text{temp3} = \text{temp1} \times \cos(\text{temp2})
\]

\[
\text{temp4} = \text{temp1} \times \sin(-\text{temp2})
\]

\[
\text{Target_disp_rows} = \text{integer} \left( \frac{\text{temp3}}{\text{Frame Rate} \times \text{Sensor Resolution}} \right)
\]

\[
\text{Target_disp_columns} = \text{integer} \left( \frac{\text{temp4}}{\text{Frame Rate} \times \text{Sensor Resolution}} \right)
\]

Sensor Frame Size

\[
\text{SensorFr_size_columns} = \text{integer} \left( \frac{(\text{Clutter Range} \times \text{Sensor Field of View} \times 96)}{(300 \times 3)} \right)
\]

\[
\text{SensorFr_size_rows} = \text{integer} \left( \frac{(\text{Clutter Range} \times \text{Sensor Field of View} \times 96)}{(300 \times 3)} \right)
\]
The sensor frame is an exact square because the horizontal and the vertical Field of View of the sensor are assumed to be same.

Sensor Frame Initial Placement

\[
\text{SensorFr}\_\text{pos\_rows} = \text{integer}\left(\frac{(\text{Clutter}\_\text{size\_rows} - \text{SensorFr}\_\text{size\_rows})}{2}\right)
\]

\[
\text{SensorFr}\_\text{pos\_columns} = \text{integer}\left(\frac{(\text{Clutter}\_\text{size\_columns} - \text{SensorFr}\_\text{size\_columns})}{2}\right)
\]

The sensor is assumed to be focused to the center of the background clutter initially.

Sensor Frame Displacement

(i) Platform Motion

\[
\text{temp} = \frac{(\text{Platform Direction} \times \pi)}{180}
\]

\[
\text{temp2} = \text{Platform Speed} \times \cos(\text{temp})
\]

\[
\text{temp3} = \text{Platform Speed} \times \sin(-\text{temp})
\]

\[
\text{SensorFr}\_\text{disp\_rows1} = \text{integer}\left(\frac{\text{temp2}}{(\text{Frame Rate} \times \text{Sensor Resolution})}\right)
\]

\[
\text{SensorFr}\_\text{disp\_columns} = \text{integer}\left(\frac{\text{temp3}}{(\text{Frame Rate} \times \text{Sensor Resolution})}\right)
\]

(ii) Sensor Scan

\[
\text{SensorFr}\_\text{disp\_rows2} = \text{integer}\left(\frac{(\text{Sensor Speed} \times \text{Clutter Range})}{(\text{Frame Rate} \times 9375)}\right)
\]

\[
\text{SensorFr}\_\text{disp\_rows} = \text{SensorFr}\_\text{disp\_rows1} + \text{SensorFr}\_\text{disp\_rows2}
\]

The sensor scan effects only the horizontal displacement of the frame.

The sensor resolution is a fixed parameter and its value is 15 meters per pixel.
The clutter size and the target signature sizes are determined by parsing the respective files and there are no equations involved.
Chapter 4. IRST Test Bench Primitives And Model Development

4.1 Introduction

As stated in section 2.1, a typical VHDL test bench consists of three major components - stimulus generator, MUT and comparator. The IRST test bench developed here at Virginia Tech also consists of these three components. The stimulus generator performs the task of generating the test stimuli, i.e., a continuous sequence of infrared image frames. The MUT embodies the IRST algorithm, and the comparator analyzes the response of the MUT to the test stimuli. Figure 4.1 shows how these three components are joined together to form a structural model of the VHDL test bench. The VHDL test bench control is comprised of the user interface, requirements interface and the test plan interface. The test plan interface is discussed in Chapter 5 and the user interface is discussed in Chapter 6.

![Figure 4.1 VHDL Test Bench for IRST algorithm.](image)

Chapter 4. IRST Test Bench Primitives and Model Development
The approach developed by Prabhakar [18] for the design of structural test benches is used for the development of the IRST test bench. The approach can be outlined as follows:

Test benches are created by schematic interconnection of the appropriate primitives from a library of primitives by the use of a commercial schematic capture tool like the Synopsys Graphical Environment (SGE) [28]. The schematic editor facility is used to interconnect the symbols created in SGE symbol editor corresponding to the different primitives. Each primitive can in turn be created as a interconnection of sub-components. The VHDL interface of SGE is used to generate the VHDL code for the test bench.

The detailed description of the implementation and working of the three components of the test bench are provided in the following sections.

4.2 Stimulus Generator

Stimulus generator is the most important component of the test bench. The structure of a typical simulation model of stimulus generator required for an IRST algorithm is described in Chapter 2. As mentioned earlier, the stimulus generator is composed of 4 sub-components - oscillator component, environment component, target component and sensor component. Figure 4.2 shows a detailed block diagram of the stimulus generator implemented as a part of this research work. The figure also shows the interaction of the stimulus generator with the VHDL Configuration and Simulation Control File, which feeds the values of secondary parameters to the sub-components. The detailed description of each of the sub-components is provided below:

Oscillator component: The oscillator component performs the task of synchronizing all the other sub-components of the stimulus generator, and controls the rate of generation of
frames. A frame is generated every clock cycle of the oscillator. The frequency of oscillation of the oscillator is dependent on one of the primary requirements, frame rate. The time for which the oscillator runs is dependent on the simulation end time parameter.

**Figure 4.2 Illustration of stimulus generator for IRST algorithm.**

*Environment component*: The output of the environment component is the image of the background clutter captured in the infrared frequency range. The clutter information can be mountains in the background, clouds or the sky. One way to generate the clutter files is by the use of a set of routines that can model the clouds, land, sea, etc. Another approach for generating this information is by the use of clutter data files captured previously. In this project the second approach is used for the generation of clutter information.
The Infrared Analysis, Measurement and Modeling Program (IRAMMP) Data Management Center (DMC) is a center established in 1988 to collect, reduce, archive and duplicate IRAMMP background clutter data for use by Naval Research Laboratories (NRL) as well as other DOD/NATO/contractor users [20]. As a member of the RASSP research contractors, Virginia Tech was provided access to the database of infrared clutter.

The advantage of using such a database is that it contains a collection of real infrared background clutter. Also, the images are reasonably big (array of size 256x256), which enables generation of sensor image frames of variable size, i.e., any size up to 256x256. Furthermore, this provides the user with a wide selection of clutter files to select from.

The environment component is designed in such a way that it always reads the contents of the clutter from a particular fixed file (CLUT.txt). Depending on the clutter file selected by the user from the database, the contents of that clutter are copied to the file CLUT.txt. Apart from the clutter file the environment component also receives as input the size of the clutter file. The requirements interface performs the task of determining the size of the clutter, i.e., the number of rows and columns of pixels in the clutter. The size parameter indicates the number of rows and columns of data to be read by the environment component from the clutter file. This feature enables the environment module to load clutter files of different sizes. Figure 4.3 shows a symbol of the environmental module created using SGE symbol editor. The figure shows the inputs and the outputs of the component.
The working of the component is as follows:

When the component receives the first active high clock signal, it reads the clutter information from the file CLUT.txt and saves the contents to a 2 dimensional variable. The number of the rows and the columns of values to be read from the file are indicated by the inputs Clutter_size_rows and Clutter_size_cols. The module then places all the values on the Clutter_information bus and generates an active high Start_T signal for the target module to start reading the clutter information from the bus. Thereafter every time the module receives an active high clock signal, it places the clutter information from the 2 dimensional variable on the Clutter_information bus and generates an active high Start_T signal. Every time the module receives an active low clock signal, it generates an active low Start_T signal.

*Target component:* The primary function of the target component is to generate a target signature, merge it with the clutter information generated by the environment component, and send the merged background information to the sensor component. One way of dealing with the problem of creating target signature is to use a software tool to generate the target signature, store it in a database of target signatures and load them when required. The target signature is a 2 dimensional array, containing gray scale intensity values of pixels representing the target image. The size of the target signature is variable.
from single point targets, i.e., size of 1 pixel, to multiple point targets of size 256x256, depending on the range of the target from the platform.

IRTool is an IRST X-Windows analysis tool developed by Arete’ Associates and NSWC/WO [26]. The IRTool can produce infrared signature images of cruise missiles. The tool takes as input the target speed, target direction of motion and the target size, and generates a 2 dimensional image of the target. We can either create a database of the targets so generated and give the user the choice of selecting the target signature from the database or generate the target signature on-line by executing the IRTool in the background with the primary parameters specified by the user. There is a tradeoff involved in creating a target signature on-line vs. using one from the database. As has been mentioned previously, IRTool generates target signatures based on the parameters specified by the user. If a target signature is created on-line, then the parameters specified by the user would be used for the creation of the target, but it takes a considerable amount of simulation time to create a target signature. Whereas, if the user selects a target signature from the database, it saves on the simulation time required to create the target.

If the user selects the target signature file from the database, the contents of the particular target signature are copied to a specific file (target.txt) which is accessed by the target component to load the target signature. The user can also select to create the target signature on-line. In that case an IRTool execution script file is generated on-line with the primary parameters supplied by the user. The IRTool software is executed in background to generate the target signature. The contents of the generated target signature are copied to the specific file (target.txt). The requirements interface loads the target signature file to determine the number of the rows and columns in the file, i.e., the size of the target. The size parameter is then input to the target component, so that the component knows how many values to read from the file. Figure 4.4 shows a symbol of the target component showing its inputs and the outputs.
The outline of the component's function is as follows:

When the component receives the first active high clock signal, it reads the target information from the file target.txt and saves the contents to a 2 dimensional variable. The number of the rows and the columns of values to be read from the file are indicated by the inputs Target_size_rows and Target_size_cols. Every time the module receives an active high Start_T signal, it starts reading the clutter information from the Clutter_information bus. After reading the clutter information from the bus the component merges the target information with the clutter information based on the target initial placement and the target displacement. The target initial placement and the target displacement in terms of rows and columns is supplied by the requirements interface. The component then places the merged clutter information on the clutter+target bus and generates an active high Start_S signal for the sensor component to start reading the complete background.
information. Every time the component receives an active low clock signal it generates an active low Start_S signal.

*Sensor component*: The basic function of the sensor component is to capture a part of the merged clutter information, i.e., clutter with target merged in it, as a sensor frame, inject errors (noise or sensor bias) and send the frame to the MUT for processing or save it for later processing by the MUT. Figure 4.5 shows a symbol of the sensor component showing the inputs and the outputs of the module.

![Diagram](image)

**Figure 4.5 The Sensor Component**

The outline of the modules function is as follows:

Every time the module receives an active high Start_S signal, it reads the merged clutter information from the Clutter+target bus. The size of the sensor frame is specified by the signals SensorFr_No_Columns and SensorFr_No_Rows. The sensor component picks up
a slice of specified size from the merged target and clutter information. The actual position of the slice in the merged target and clutter is specified by the signals SensorFr_Pos_Rows, SensorFr_Pos_Columns, SensorFr_disp_Hz and SensorFr_disp_Vt signals.

Once the sensor frame has been captured from the merged clutter information, the component varies the intensity values of the pixels of the sensor frame to inject noise or sensor bias. Three different functions have been developed to model these errors. The different kinds of errors possible are explained in Chapter 3. The actual characteristic of the variation and the level of variation to be injected are specified by the parameters error type and error level.

The user can also select to inject another type of error, uneven offset. The characteristic of this error is discussed in Chapter 3. This function models an uneven gain of the sensor across the sensor frame. The function has been so designed that by making the value of the parameter, sensor gain variation \(0\), the effect of this function can be nullified. This feature enables the use of same sensor model to characterize both kinds of sensors, i.e., ones that introduce this kind of error and also those which do not. This is an important consideration in the design of reusable models.

4.3 Comparator

The stimulus generator generates the test vectors. The MUT then operates on the test vectors. Once the response of the MUT to the test vector is available there must be a way to determine the correctness of the response. One approach is to manually go through the results and determine this. This would be laborious and error prone. It would be better and more convenient if the test bench itself can generate the expected response to the test
vectors generated and then compare it with the response of the MUT to verify correctness.

The comparator is a component of the test bench which performs this task. It receives as input the expected response from the stimulus generator and the simulation results from the MUT and then compares them to decide whether the response of the MUT is correct or not.

The IRST MUT performs the following functions:
(i) Identify the motion of the sensor frame in pixels.
(ii) Identify the motion of the target signature in pixels.
(iii) and identify the target signature.

The expected sensor frame displacement is the value of the signals SensorFr_disp_Hz and SensorFr_disp_Vt, which are inputs to the sensor component of the stimulus generator. The expected target signature displacement in pixels is the value of the signals Target_disp_Hz and Target_disp_Vt, which are inputs to the target module of the stimulus generator. The expected target signature is just the contents of the target signature file selected from the database. The MUT response comprises the sensor frame displacement (no of rows and columns that the second frame is displaced relative to first frame), target frame displacement (no of rows and columns that the target is displaced in successive frames), and the detected target image (2 dimensional array of target image). The comparator compares the sensor frame displacement and the target displacement reported by the MUT with the corresponding expected responses for an exact match. In the case of the target signature the comparator computes the mean square error (MSE) between the average pixel intensity of the target detected by the MUT and the actual target signature selected to create the sensor frames. If this value of MSE is less than 0.1.
i.e., 90 percent match, the target signature detected by the MUT is considered correct. This metric works well in determining the correctness of the target signature detected.

4.4 IRST MUT Prototype

As has been indicated in section 2.3, the basic task of the IRST algorithm is to detect the displacement of the successive sensor frames caused by a combined effect of sensor scan action and the platform motion, detect the displacement of the target in successive frames due to target motion, and identify the signature of the target.

The algorithm performs the above mentioned functions in the order stated. The ability of the algorithm to identify the correct target motion is dependent on the ability of the algorithm to detect correct sensor frame displacement in successive frames. Also, The correct detection of the target signature is dependent on the proper detection of the target displacement in successive frames.

The actual process involved in the detection of sensor frame displacement or target displacement or target signature is to find the best alignment of two successive frames (used to find sensor frame displacement) or the best alignment of slices of the successive frames (used to find target displacement and target signature). The shift corresponding to the best alignment represents the actual displacement. A metric was designed to characterize this best alignment of successive frames. The calculation of the actual value of this metric is as follows:

For every linear combination of the alignments of one frame on the other, the differences in the intensity values of the individual pixels are summed up. This sum is then divided by the total number of corresponding pixels in the overlapping part have different
pixel intensity values. The resultant value is the value of the metric. If the number of pixels that do not match is zero, i.e., an exact match, the value of the metric is set to zero.

The alignment with the least value of this metric represents the best alignment. The shift corresponding to this alignment represents the actual displacement. This metric was found to give accurate results in most situations. It gives wrong values of shifts when two different alignments have the same value of the metric or in the case of excessive noise injection.

The amount of shifts that can be made in order to align the successive frames is characterized by a parameter called ScanRange. The algorithm tries a linear combination of shifts from -ScanRange to +ScanRange in both horizontal and vertical directions. Therefore, the algorithm can detect a maximum sensor frame displacement of ScanRange pixels in both horizontal and vertical directions.

Figure 4.6 shows the working of the IRST algorithm. Figures 4.6(a) and (b) show two successive frames. Figure 4.6(c) shows the best alignment of the two frames, i.e., from the figure the second frame has moved one pixel in the vertical direction and one pixel in the negative horizontal direction. Figure 4.6(d) shows the difference array, i.e., the array containing the difference of the intensity values in the overlapping part of the frames. If the sensor shift found using the metric explained above is indeed the right displacement of the frame then except for the shaded region shown in figure 4.6(d) the remaining values of the array cancel out. The rectangle enclosing the complete shaded region in the difference array is characterized as the area of target concentration as is shown in figure 4.6(e). The slice comprising the area of concentration is then picked up from both the individual frames, as is shown in figures 4.6(f) and (g).
The target displacement is found by finding the best alignment of the areas of concentration (frame slices) of the two frames. The maximum amount of linear shifts possible to find the target displacement is characterized by a model parameter called target_scan_range. The process and the metric used are similar to the one used to find the sensor frame shift. Figure 4.6(h) shows the best alignment of the slices, i.e., from Figure 4.6(h) the target has moved one pixel in the negative vertical direction and one pixel in the horizontal direction. If the target displacement found using the metric is indeed the actual target displacement, then the shaded part is the actual target signature as is shown in the Figure 4.6(i). In the presence of noise the algorithm picks up parts of clutter surrounding the target also as a part of the target.
This algorithm was found to perform very consistently in different background clutters and also with different target signatures.
Chapter 5. Test Plan Interface And Iterative Testing

5.1 Introduction

As mentioned above, an environment based test bench has the ability to model the environment surrounding the MUT. But in order to test a particular operation of the MUT it is essential to configure the test bench so that the appropriate test stimuli are generated. A main emphasis of this thesis is to develop an approach to configure the test bench according to a particular testing strategy. With such a feature the user would be able to select a particular testing scenario and the test bench system would then configure the test bench to generate the corresponding test vectors which would facilitate the testing of the MUT according to the selected testing strategy. This would relieve the user of manually configuring the test bench with appropriate VHDL models and input parameters.

A test plan interface was developed as a part of this thesis to provide a test bench with the above feature. A test plan organizes the system requirements in terms of how these requirements will be tested [27]. It divides the requirements into groups and allocates a set of tests (termed as a test group) to each of the requirement groups. Once the user selects a particular test group, the test plan configures the test bench accordingly.
The test bench is then simulated to generate the appropriate test stimuli and to execute the MUT with the generated stimuli. This approach relates to a test bench system as follows:

A structural test bench is formed by the interconnection of primitive components. In VHDL this interconnection is formed in a two stage process [27]. First the interconnection of a set of primitive entities is specified. Next each primitive entity is bound to a library model of the primitive. These bindings can take place within the structural model itself, or they can be placed in a separate VHDL description called a configuration [3]. The bindings select which VHDL model to use.

The bindings of these primitives to the respective VHDL models in the library is explained below with the help of an example VHDL configuration description.

Configuration TP1 of IRST_StimGen is

for structural

for all: target_primitive use entity PrimLib.target_model1;
end for;

for all: oscillator_primitive use entity PrimLib.osc_model1;
end for;

for all: environment_primitive use entity PrimLib.environment_model1;
end for;

for all: sensor_primitive use entity PrimLib.sensor_model1;
end for;
end for;
end TP1;

In the above configuration description ‘TP1’ refers to the name of the configuration and ‘IRST_StimGen’ refers to the name of the structural VHDL model being built. Each ‘for all’ construct of the configuration description describes the primitive and
the actual VHDL model in the library that it is to be bound to (e.g., the first ‘for all’
construct describes that every instance of the primitive ‘target primitive’ in the structural
model is to be bound with the VHDL model ‘target model’ from the library ‘PrimLib’).

In an ideal case, depending on the actual binding required for the particular test
group the corresponding configuration description would be generated automatically
online by the test plan interface. Another approach is to prepare the configuration
descriptions beforehand and store them. Depending on the test group selected by the user
the corresponding configuration description can be selected.

The test plan interface also prompts the user to specify a set of primary parameters
required for the creation of the test stimuli. The remaining primary parameters are
specified by the test plan itself. Primary parameters specified by the user and the test plan
are then fed to the requirements interface for the calculation of the secondary parameters
required by the test bench. A simulation control file is then generated online to control the
simulation of the VHDL test bench and also to input the values of secondary parameters
into the test bench at the appropriate simulation time. 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. Figure 3.1 depicts how a test plan interface
fits into a high level test bench generation system.
Figure 5.1 A high level test bench generation system with test plan interface.

5.2 Test plan Interface for IRST test bench

As an illustration of the above approach a test plan interface was developed for IRST test bench with four test groups in it. Chapter 4 discusses the development of the VHDL models of the test bench primitives. The interconnection of the primitives for the IRST test bench and the stimulus generator is also described in Chapter 4. The present version of the primitives library of IRST primitives consists of a single VHDL model of each primitive. Therefore every test group in the test plan maps to the same bindings of the test bench primitives. As all the test plans use the same bindings, the configuration description of the test bench was included in the structural model of the test bench itself i.e., the component binding is not done on-line. Once the library of primitives is populated with different models of each primitive and an approach for automatic configuration generation has been developed, the test plan interface can be enhanced to generate the required configuration description on-line.
In the present version of the test plan interface, the primary requirements of the test bench are divided into two groups. This grouping of requirements is different for each test group in the test plan interface. The first set of requirements are specified by the user or they are defaults from the specifications repository. The actual requirements to be input by the user are prompted by the test plan interface. The values of the remaining parameters are specified by the test plan interface itself. A detailed description of the test groups is given below.

1. **Target displacement detection**: Every IRST algorithm has a bound on the amount of target displacement in successive frames that the model can detect. If the amount of displacement in successive frames is more than the particular bound, the MUT cannot detect the proper target signature or the correct target motion. The parameters that affect the displacement of the target in successive frames are speed of the target, target range, frame rate and clutter range. The requirements grouping for this test group is shown in the table below.

<table>
<thead>
<tr>
<th>Requirements specified by user</th>
<th>Target Speed</th>
<th>Target Range</th>
<th>Clutter Range</th>
<th>Frame Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements specified by test plan interface itself</td>
<td>No of Frames</td>
<td>Field of View</td>
<td>Platform Type</td>
<td>Platform Speed</td>
</tr>
<tr>
<td></td>
<td>Platform Direction</td>
<td>Sensor Type</td>
<td>Sensor Scan Rate</td>
<td>Target Initial Position</td>
</tr>
<tr>
<td></td>
<td>Target Direction</td>
<td>Error Type</td>
<td>Error Level</td>
<td>Sensor Gain Variation</td>
</tr>
</tbody>
</table>
The aim of the tests in this test group is to create test vectors to enable the proper evaluation of the ability of the MUT to detect proper target displacement in successive frames. Therefore the test bench is configured to generate test vectors with standing platform, i.e., no platform motion, staring sensor, i.e., no sensor scan, and no noise injection.

2. *Sensor frame displacement detection*: As has been discussed in Chapter 4, the proper detection of the target motion and the target signature by the IRST algorithm is dependent on its ability to detect the movement of the sensor frame in successive frames of the test vector. As has been mentioned before, this displacement is caused by both platform motion and sensor scan motion. It is necessary to find this displacement because the target displacement in successive frames is actually the relative displacement between the frame displacement and target actual displacement. Therefore, to properly track and detect a target it is very critical to find the correct sensor frame displacement. Every IRST algorithm has a bound on the amount of displacement in the successive sensor frames that it can detect.

The parameters that affect the displacement of the sensor frame in successive frames of the test stimuli are the speed of the platform, the sensor scan speed, the frame rate and the clutter range. The requirements grouping for this test group is shown in Table 5.2.
### Table 5.2 Requirements Grouping for Test Group 2.

| Requirements specified by user | Platform Type  
|                              | Platform Speed  
|                              | Sensor Type  
|                              | Sensor Scan Rate  
|                              | Clutter Range  
|                              | Frame Rate  
| Requirements specified by Test Plan itself | No of Frames  
|                              | Field of View  
|                              | Platform Direction  
|                              | Target Speed  
|                              | Target Initial Position  
|                              | Target Direction  
|                              | Target Range  
|                              | Error Type  
|                              | Error Level  
|                              | Sensor Gain Variation  

The aim of the tests in this test group is to create test vectors to enable the proper evaluation of the ability of the MUT to detect the frame displacement in successive frames. Therefore the test bench is configured to generate test vectors with minimum target displacement and no noise injection.

3. *Noise sensitivity determination*: Every system has some sensitivity to noise. If this injected noise is beyond a particular limit, the model can no longer perform as expected. In the presence of excessive amounts of noise, the IRST MUT fails to detect the proper frame displacement and/or proper target displacement. It is therefore of prime importance to determine the sensitivity of the MUT to noise.

The parameters that affect the amount of noise injected into the system are error type, error level, sensor gain variation. The requirements grouping for this test group of the test plan interface is shown in the table below.
Table 5.3 Requirements Grouping for Test Group 3.

<table>
<thead>
<tr>
<th>Requirements specified by user</th>
<th>Error Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Error Level</td>
</tr>
<tr>
<td></td>
<td>Sensor Gain Variation</td>
</tr>
<tr>
<td>Requirements specified by Test Plan itself</td>
<td>Frame Rate</td>
</tr>
<tr>
<td></td>
<td>No of Frames</td>
</tr>
<tr>
<td></td>
<td>Field of View</td>
</tr>
<tr>
<td></td>
<td>Clutter Range</td>
</tr>
<tr>
<td></td>
<td>Target Range</td>
</tr>
<tr>
<td></td>
<td>Platform Type</td>
</tr>
<tr>
<td></td>
<td>Platform Speed</td>
</tr>
<tr>
<td></td>
<td>Platform Direction</td>
</tr>
<tr>
<td></td>
<td>Sensor Type</td>
</tr>
<tr>
<td></td>
<td>Sensor Scan Rate</td>
</tr>
<tr>
<td></td>
<td>Target Speed</td>
</tr>
<tr>
<td></td>
<td>Target Initial Position</td>
</tr>
<tr>
<td></td>
<td>Target Direction</td>
</tr>
</tbody>
</table>

As the main aim of this test case is to evaluate the noise sensitivity of the model, the actual value of the other parameters specified by the test plan interface should be such that MUT would perform well with the test vectors created using those values of parameters in the absence of noise. In such a setup any improper response of the MUT would be caused only by the injected noise. This would facilitate the evaluation of the sensitivity of MUT to noise.

4. General test group: Apart from the three test groups described above a special test group was developed to allow the user to specify all the test bench parameters in order to create tests which do not fall into any of the three groups. In this test case the test plan interface prompts the user to specify all the primary parameters and feeds them directly to the requirements interface for calculation of secondary parameters and to be fed into the VHDL model.
Apart from the generation of these test vectors the stimulus generator also provides the expected response to the comparator. The response of the MUT is also fed to the comparator to be compared with the expected response. The match between the two indicates the proper functioning of the MUT.

5.3 Iterative Testing

The test plan interface specifies the value of the specification parameter to be used for the creation of the test frames. Many times it is necessary to find the limiting value for a particular specification parameter, i.e., the value of the parameter beyond which the MUT fails to perform appropriately. For example, the user might want to know for which value of target speed in the target displacement detection test group the MUT fails to detect the proper target displacement. This can be achieved by an iterative search for the limiting value. This search involves repeated execution of the test bench generation and simulation software.

The iterative search procedure could be either a gradual increase of the value of the parameter by some specified increment or by a binary search for the value in the range of values that the parameter can be assigned. This process is continued until the MUT fails. The maximum value of the parameter for which the MUT performs appropriately is the limiting value of the particular parameter. This mode of test bench operation corresponds to the on-line, adaptive test bench configuration described in Chapter 2.

In order to illustrate this the test plan interface was provided with the facility to find the limiting value of one parameter of the user specified requirements in each of the first three test groups in the test plan interface.
The table below shows the parameters and the corresponding test group for which the iterative search procedure has been automated in the present version of the test bench system.

<table>
<thead>
<tr>
<th>Test Group</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target displacement detection</td>
<td>Target Speed</td>
</tr>
<tr>
<td>Sensor frame displacement detection</td>
<td>Platform Speed</td>
</tr>
<tr>
<td>Noise Sensitivity determination</td>
<td>Noise Level</td>
</tr>
</tbody>
</table>

The procedure used for iteration is to gradually increment the parameter every time the MUT response is the same as the expected response. The actual value of the increment is specified by the user. The limiting value of the other parameters in each test group can be found in this way. The limiting value of the parameters obtained for the above parameters using the iterative mode of testing for the IRST MUT developed here at Virginia Tech is provided in Chapter 7.
Chapter 6. System Integration And User Interface Development

6.1 System Integration

Chapters 3, 4 and 5 of this thesis describe the development of pieces of IRST test bench. The methodology used for the development of these pieces is a very high level approach and involves the use of different computer tools and programming languages. For example, the simulation models of the test bench primitives were created in VHDL using SGE, the requirements interface and the test plan interface were designed in C, sensor frame display function were coded in Matlab, templates of VHDL simulation control file were created using Synopsys script language, target signatures were generated using IRTool, the IRTool execution script was written in C shell language and the specifications repository was developed using SGE. It is very important to integrate all these different pieces of the test bench in a seamless fashion to form an automated test bench generation system.

Figure 6.1 shows the system level diagram of the IRST test bench generation system developed here at Virginia Tech.
A C program was developed to integrate these different modules of test bench system. This C program provides a single platform for the control and execution of these different modules. The program also performs the required format translation of data produced by one module for use by other modules. For example, the IRTool generates the signature of targets in the HDF graphical format [30], whereas the VHDL models require the target signature in the ASCII format.
6.2 User Interface Development

A menu driven X-Windows user interface, TBGUI (Test Bench Generator User Interface), was developed to provide the user with a friendly environment to operate test bench generation system for the creation of the test bench according to the test to be performed by the user. TBGUI also controls the simulation of the created test bench and the display of the simulation results during or after the simulation.

TBGUI was developed in C using X-View layer libraries of the X-11 protocol. The user interface provides the user with a set of menus that lead the user through the process of test bench configuration and simulation. The user interface either prompts the user with a list of possible selections, or asks the user to key in input values in a certain range. Figure 6.2 shows the overall menu structure provided by the user interface. It also shows the flow of the selections to be made by the user.

The set of menus provided by the user interface are:

1. *Stimulus Source*: This menu allows the user to select the mode of configuration. The test bench can be operated in two modes, file I/O or code-generated, which basically describe the source of the test vectors. In the file I/O mode the test vectors are loaded from a specified file in the test vector database and sent to the MUT for processing. This mode of operation is similar to the off-line test bench configuration described in Chapter 2. In the other mode of operation, code-generated, the test vectors are created by the on-line creation and simulation of the test bench. The generated test vectors can either be saved for later simulation of the MUT in off-line configuration or sent to the MUT for on-line processing, i.e., in the same simulation.
2. *Test Plan*: This menu allows the user to select any one of the four test groups available in the test plan interface. The purpose of this selection is different for the file I/O mode and code-generated mode. In the file I/O mode the user selects the test plan to choose a test vector created previously using the particular test plan, whereas in the code-generated mode the user selects the test plan to create the test vectors using the particular test plan.
3. **Select Input parameter values:** Depending on the test plan selected, the user is prompted to enter the values of the parameters corresponding to the test group required for the creation of the test vector. The appropriate range of values for each parameter is also specified. This menu selection is available only with the code-generated option of *stimulus source* menu.

4. **Test Case:** This menu selection allows the user to select a particular test vector from the database of test vectors created previously using the same test plan selected by the user for the present simulation. This menu selection is available only with the file I/O option of *stimulus source* menu.

5. **Test Mode:** This option allows the user to either perform a single test or execute the test bench generation system iteratively to find the limiting value of a particular parameter. The iterative test mode is described in Chapter 5. This menu selection is available only with the code-generated option of *stimulus source* menu.

6. **Clutter File Selection:** This selection provides the user with a list of clutter files in the database to select from for the creation of test vectors. This selection is available only with the code-generated option of the stimulus source menu selection.

7. **Target Type:** This menu allows the user to select either a single point target or a target signature generated by IRTool. As mentioned before the target signatures generated by IRTool are generally multi-pixel images. This menu selection is available only with the code-generated version of test bench configuration.

8. **IRTool Target Selection:** If the user selects the IRTool target type. Then there are two options possible. The user can either select the target from the database of the target
signatures created previously using IRTool, or select to create a signature on-line by executing IRTool in the background. If the user chooses to select a target signature from the database of target signature then he is presented with a list of target signatures available in the database of IRTool generated signatures to select from.

9. Simulation Menu: This option allows the user to select either to simulate only the stimulus generator or the complete test bench. If the user chooses to simulate only the stimulus generator then the test bench generation system executes to configure the stimulus generator and it is simulated to create the required test vectors. The generated test vectors can be saved to the database of test vectors for later execution of the test bench in the file I/O configuration. If the user selects to simulate the complete test bench then the test vectors are generated on-line and are sent to the MUT for processing in the same simulation. This option is available only with code-generated version of the test bench configuration.

10. MUT Simulation: This option is available only if the user selects to simulate the complete test bench. The user can simulate the MUT in two different ways, either on-line or post. In the on-line mode of operation, as the stimulus generator creates each frame, it is sent to the MUT for processing. In the post mode of MUT simulation the stimulus generator first generates all the test frames and then sends all the frames to the MUT for processing.

Based on the options selected by the user the corresponding test bench, test data files and the simulation control files are generated. The test bench is then simulated to perform the selected test or just generate the appropriate test frames.
Chapter 7. Results

The Synopsys VHDL simulator [33] was used for the simulation of the test benches created using our test bench generator system.

6.1 Simulation results of file I/O mode

As has been mentioned before, in the file I/O mode the user selects an already created test vector from the data-base and the MUT is executed with the selected test vector. Figure 7.1 shows a test vector selected for a file I/O test case. The test vector consists of a continuous sequence of 4 frames. Figure 7.2 shows the output produced by the MUT when exercised with the selected test vector. The figure indicates that the MUT detected a sensor frame motion of 1 pixel in the horizontal direction, -2 pixels in the vertical direction, i.e., in the downward direction, target movement of three pixels in horizontal direction and -3 pixels in the vertical direction. It also shows the target signature detected by the MUT. Figure 7.3 shows the output of the comparator. The figure shows the target speed, platform speed, the noise level used to create the test vectors and also that the response of the MUT was correct.
Figure 7.1 Test vectors used in the file I/O test case.
Figure 7.2 MUT outputs for the file I/O test case.

Chapter 7. Results
6.2 Simulation results of code-generated mode

As has been mentioned previously, in the code-generated mode of test bench configuration, the test vectors are generated online and the MUT is also executed with the generated test vectors in the same simulation. Simulation results of the test bench configured using noise sensitivity test group of the test plan interface are illustrated here. Figure 7.4 shows the clutter image selected from the database for the creation of the test vectors. Figure 7.5 shows the test vector created by the test bench. The test vector consists of a continuous series of four sensor frames. Figure 7.6 shows the output of the MUT to the test vector. It shows that the MUT detected a frame movement of 1 pixel in the negative horizontal direction, 1 pixel in the vertical direction, target movement of 1 pixel movement in the horizontal direction and 1 pixel in the negative vertical direction. Figure 7.7 shows the outputs of the comparator. It shows the noise level injected into the sensor frames and that the response of the MUT was correct.
Figure 7.4 Clutter selected for code-generated test case.
Figure 7.5 Test vector generated in the code-generated test case.
Figure 7.6 MUT outputs for the code-generated test case.
6.3 Results of Iterative Test Mode

The test bench generation system was operated in the iterative mode to find the limiting value of a set of parameters for the IRST algorithm developed here at Virginia Tech. As stated before, the iteration procedure used was - repeated incrementing of the parameter by increments specified by the user until the MUT fails to respond correctly. Table 7.1 shows the limiting values found for the parameters mentioned in Chapter 5.

<table>
<thead>
<tr>
<th>Test Group</th>
<th>Parameter</th>
<th>Increment</th>
<th>Limiting Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target displacement detection</td>
<td>Target Speed</td>
<td>25 m/s</td>
<td>450 m/s</td>
</tr>
<tr>
<td>Sensor frame displacement</td>
<td>Platform Speed</td>
<td>15 m/s</td>
<td>225 m/s</td>
</tr>
<tr>
<td>Noise sensitivity</td>
<td>Noise Level</td>
<td>10</td>
<td>110</td>
</tr>
</tbody>
</table>
The table shows that the target speed increments of 25 m/s was selected. The MUT failed to give a correct response for the test vectors created with target speed value of 475 m/s. Therefore the limiting value of target speed was reported as 450 m/s. The actual value could be any value in the range 451 m/s - 475 m/s. Therefore the precision of the limiting value found is dependent on the increments selected by the user. In the table the noise level increments and limiting value of noise level does not have any units because the noise is just the offset in the intensity of the pixels of the infrared image.
Chapter 8. Conclusions And Future Work

This thesis has presented a methodology to develop an integrated test bench generation system systematically and developed an approach to configure the test bench according to a particular testing strategy selected by the user. As an illustration of this a test bench system was developed for the Infrared Search and Track algorithm. The VHDL primitives of the test bench and the prototype IRST MUT were developed based on the work done previously here at Virginia Tech. A test plan interface was developed to configure the test bench according to a test plan. The individual modules of the test bench generation system were integrated into a single automated test bench generator system. An X-Windows interface was developed to facilitate easy operation of the test bench generator system.

Some of the avenues in which future work is possible are as follows:

The present version of the IRST primitives library contains only single VHDL models for each test bench primitive. The primitives library can be populated by developing multiple VHDL models for each primitive. These models can have different architecture or same architecture but designed at different abstraction levels.

The present version of the test plan interface has only 4 test groups; other test groups can be developed. Also, the limiting value of only one parameter of each test group
can be found automatically in the present version. The system can be enhanced to find the limiting values of every parameter.

The X-Windows user interface was developed using C libraries available at the X-View layer of the X11 protocol. This interface can be developed using libraries available at a lower level of X11 protocol that provide access to more primitive functions. This would facilitate development of a more robust user interface.

The specification repository is presently implemented using SGE, a new specification repository can be developed using X-Windows programs that would provide a better means for parameter verification, extraction and would also facilitate using physical data types to represent system parameters.
Bibliography


Bibliography


Appendix A: VHDL Code listing

Infrared Search and Track Primitives

Oscillator Component

entity OSC is

   Port ( PER : Time;
         INIT : In BIT;
         CLK : Out BIT );

end OSC;

architecture BEHAVIORAL of OSC is

begin

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

end BEHAVIORAL;
use work.all;
use STD.TEXTIO.all;
use work.USER_TYPES1.all;

entity ENVIRON is
  Port ( DIM_I : IN INTEGER;
         DIM_J : IN INTEGER;
         CLK : IN BIT;
         START_T : OUT BIT);
end ENVIRON;

architecture BEHAVIORAL of ENVIRON is
signal DONE : BIT;

begin
  process(CLK)
  constant Space : string:= " ";
  variable VLINE : LINE;
  variable COUNT : INTEGER:=1;
  variable COUNT1 : INTEGER:=999;
  variable I : INTEGER:=0;
  variable J : INTEGER:=0;
  variable TEMP : INTEGER:=0;
  variable CLUT_TEMP:INT_2D(0 to 255, 0 to 255):=(others =>(others=> 0));
  file INCLUTTER :TEXT is in "clutter.txt";
  file OUTCLUTTER : TEXT is out "clutter_bck.txt";
  begin

    if (CLK'event and CLK = '1') then
      if DONE = '0' then
        for J in 0 to (DIM_J-1) loop
          READLINE(INCLUTTER,VLINE);
          for I in 0 to (DIM_I-1) loop
            READ(VLINE, TEMP);
            CLUT_TEMP(J,I) := TEMP;
          end loop;
        end loop;

      end if;
    end if;
  end process;
end BEHAVIORAL;
end loop;
DONE<= '1';
end if;
WRITE(VLINE,COUNT);
WRITELINE(OUTCLUTTER,VLINE);
for J in 0 to (DIM_J-1) loop
  for I in 0 to (DIM_I-1) loop
    TEMP := CLUT_TEMP(J,I);
    WRITE(VLINE, TEMP);
    WRITE(VLINE, Space);
  end loop;
  WRITELINE(OUTCLUTTER,VLINE);
end loop;
COUNT := COUNT + 1;
START_T<= '1';
end if;
if (CLK’event and CLK = '0') then
  START_T<= '0';
end if;
end process;
end BEHAVIORAL;

**Target Component**

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

entity TARGET is
  Port(
    DIM_I  : IN INTEGER;
    DIM_J  : IN INTEGER;
    SIZE_X : IN INTEGER;
    SIZE_Y : IN INTEGER;
    INIT_X : IN INTEGER;
    INIT_Y : IN INTEGER;
    DISP_X : IN INTEGER;
    DISP_Y : IN INTEGER;
    START_T : IN BIT;
  )
START_S : OUT BIT);
end TARGET;

architecture BEHAVIORAL of TARGET is
signal DONE:BIT;

begin
  process(START_T)
  constant Space : STRING:=" ";
  variable I,J,K,L : INTEGER;
  variable trigger : BIT;
  variable temp : INTEGER:=0;
  variable VLINE : LINE;
  variable INITx : INTEGER;
  variable INITy : INTEGER;
  variable COUNT : INTEGER:=1;
  variable COUNT1 : INTEGER:= 999;
  variable target :INT_2D(0 to 8, 0 to 8):=(others=>(others=>0));
  variable back_grn : INTEGER := 0;
  variable CLUT_TEMP:INT_2D(0 to 255, 0 to 255):=(others=>(others=>0));
  file INCLUT : TEXT is in "clutter_bck.txt";
  file TargSig : TEXT is in "target.txt";
  file TARGIM : TEXT is out "targim.txt";

  begin
    if (START_T'event and START_T = '1') then
      if DONE= '0' then
        for J in 0 to (SIZE_Y-1) loop
          READLINE(TargSig, VLINE);
          for I in 0 to (SIZE_X-1) loop
            READ(VLINE, temp);
            target(J, I) := temp;
          end loop;
        end loop;

        INITx := INIT_X;
        INITy := INIT_Y;
        DONE <= '1';
      end if;
      READLINE(INCLUT,VLINE);
      READ(VLINE, TEMP);
      if (TEMP = COUNT) then
        for J in 0 to (DIM_J-1) loop
          if (COUNT1 = COUNT) then
            
          end loop;
        end if;
      end if;
    end if;
  end process;
end BEHAVIORAL;
READLINE(INCLUT, VLINE);
for I in 0 to (DIM_I-1) loop
  READ(VLINE, TEMP);
  CLUT_TEMP(J,I) := TEMP;
end loop;
end loop;
end if;
K := 0;
for J in INITy to (INITy + SIZE_Y -1) loop
  if((INITy >= 0) and (INITy + SIZE_Y <= DIM_J)) then
    L := 0;
    for I in INITx to (INITx + SIZE_X -1) loop
      if((INITx >= 0) and (INITx + SIZE_X <= DIM_I)) then
        back_grn := CLUT_TEMP(J, I);
        if (back_grn < targt(K,L)) then
          CLUT_TEMP(J, I) := targt(K, L);
        end if;
      end if;
      L := L+1;
    end loop;
  end if;
  K := K+1;
end loop;
INITx := INITx + DISP_X;
INITy := INITy + DISP_Y;
WRITE(VLINE,COUNT);
WritelINe(TARGIM,VLINE);
for J in 0 to (DIM_J-1) loop
  for I in 0 to (DIM_I-1) loop
    temp := CLUT_TEMP(J,I);
    WRITE(VLINE, temp);
    WRITE(VLINE, Space);
  end loop;
  WritelINe(TARGIM, VLINE);
end loop;
COUNT := COUNT + 1;
START_S <= '1';
end if;
if (START_T'event and START_T = '0') then
  START_S <= '0';
end if;
end BEHAVIORAL;

Sensor Componant

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

entity SENSOR is
  Port ( DIM_I : IN INTEGER;
      DIM_J : IN INTEGER;
      SIZE_X : IN INTEGER;
      SIZE_Y : IN INTEGER;
      INIT_X : IN INTEGER;
      INIT_Y : IN INTEGER;
      DISP_X : IN INTEGER;
      DISP_Y : IN INTEGER;
      NoiVal : IN INTEGER;
      NoiTyp : IN INTEGER;
      SenGain : IN INTEGER;
      START_S : IN BIT;
      START_MUT : OUT BIT);
end SENSOR;

architecture BEHAVIORAL of SENSOR is
signal DONE:BIT;

begin
  process (START_S)
    constant Space : STRING := " ";
    variable I : INTEGER := 0;
    variable J : INTEGER := 0;
    variable COUNT : INTEGER := 1;
    variable COUNT1 : INTEGER := 999;
    variable K,L : INTEGER := 0;
    variable INITx : INTEGER;
    variable INITy : INTEGER;
    variable IMAGE_TEMP:INT_2D(0 to 255,0 to 255):=(others=>(others=>0));
variable CLUT_TEMP :INT_2D(0 to 255,0 to 255):=(others=>(others=>0));
variable VLINE : LINE;
variable TEMP : INTEGER:= 0;
file IMAGES : TEXT is out "frames.txt";
file INCLUT : TEXT is in "targim.txt";

begin
  if (START_S='event and START_S = '1') then
    if (DONE = '0') then
      DONE <= '1';
      INITx:= INIT_X;
      INITy:= INIT_Y;
    end if;
    READLINE(INCLUT,VLINE);
    READ(VLINE, TEMP);
    if (TEMP = COUNT) then
      for J in 0 to (DIM_J-1) loop
        READLINE(INCLUT,VLINE);
        for I in 0 to (DIM_I-1) loop
          READ(VLINE, TEMP);
          CLUT_TEMP(J,I) := TEMP;
        end loop;
      end loop;
    end if;
    if (INITx< 0) then
      INITx := 0;
    end if;
    if (INITx + SIZE_X > DIM_J) then
      INITx := DIM_J - SIZE_X;
    end if;
    if (INITy < 0) then
      INITy := 0;
    end if;
    if (INITy + SIZE_Y > DIM_J) then
      INITy := DIM_J - SIZE_Y;
    end if;
    if ((INITx+SIZE_X < DIM_J) and (INITy+SIZE_Y < DIM_J)) then
      K := 0;
      for J in INITy to (INITy + SIZE_Y - 1) loop
        L:= 0;
        for I in INITx to (INITx + SIZE_X - 1) loop
          IMAGE_TEMP(K, L) := CLUT_TEMP(J, I);
          L:= L+1;
        end loop;
      end loop;
    end if;
end loop;
K := K + 1;
end loop;
end if;
if (SenGain /= 0) then
SensorGain(SenGain, SIZE_X, SIZE_Y, IMAGE_TEMP);
end if;
if (NoiVal /= 0) then
  if (NoiTyp = 0) then
    ConsNoise(NoiVal, SIZE_X, SIZE_Y, IMAGE_TEMP);
  elsif (NoiTyp = 1) then
    RelNoise(NoiVal, SIZE_X, SIZE_Y, IMAGE_TEMP);
  elsif (NoiTyp = 2) then
    RandNoise(NoiVal, SIZE_X, SIZE_Y, IMAGE_TEMP);
  end if;
end if;
INITX := INITX + DISP_X;
INITY := INITY + DISP_Y;
WRITE(VLINE,COUNT);
WRITELINE(IMAGES,VLINE);
for J in 0 to K-1 loop
  for I in 0 to L-1 loop
    WRITE(VLINE,IMAGE_TEMP(J,I));
    WRITE(VLINE,Space);
  end loop;
  WRITELINE(IMAGES,VLINE);
end loop;
COUNT := COUNT + 1;
START_MUT <= '1';
end if;
if (START_S'event and START_S = '0') then
  START_MUT <= '0';
end if;
end process;
end BEHAVIORAL;

**Noise Function**

use work.USER_TYPES1.all;
use work.MATH_REAL.all;

Appendix A
use STD.TEXTIO.all;

package NoiseFunc is

procedure ConsNoise(cons: in integer;
    DIM_I, DIM_J: in integer;
    CLUT: inout INT_2D(0 TO 255, 0 TO 255));

Procedure RelNoise(rel: in integer;
    DIM_I, DIM_J: in integer;
    CLUT: inout INT_2D(0 TO 255, 0 TO 255));

Procedure RandNoise(rel: in integer;
    DIM_I, DIM_J: in integer;
    CLUT: inout INT_2D(0 TO 255, 0 TO 255));

Procedure SensorGain(num: in integer;
    DIM_I, DIM_J: in integer;
    CLUT: inout INT_2D(0 to 255, 0 to 255));

end NoiseFunc;

package body NoiseFunc is

Procedure ConsNoise(cons: in integer;
    DIM_I, DIM_J: in integer;
    CLUT: inout INT_2D(0 TO 255, 0 TO 255)) is
begin
    for J in 0 to DIM_J-1 loop
        for I in 0 to DIM_I-1 loop
            CLUT(J, I) := CLUT(J, I) + cons;
            if CLUT(J, I) < 0 then
                CLUT(J, I) := 0;
            elsif (CLUT(J, I) > 255) then
                CLUT(J, I) := 255;
            end if;
        end loop;
    end loop;
end;

Procedure RelNoise(rel: in integer;
    DIM_I, DIM_J: in integer;
CLUT: inout INT_2D(0 TO 255, 0 TO 255)) is
begin
  for J in 0 to DIM_J-1 loop
    for I in 0 to DIM_I-1 loop
      CLUT(J, I) := (100 + rel)*CLUT(J,I)/100;
      if CLUT(J, I) < 0 then
        CLUT(J, I) := 0;
      elsif (CLUT(J, I) > 255) then
        CLUT(J, I) := 255;
      end if;
    end loop;
  end loop;
end;

Procedure RandNoise(rel: in integer; DIM_J, DIM_I: in integer; CLUT: inout INT_2D(0 TO 255, 0 TO 255)) is
begin
  variable DIM_I1 : integer := 10;
  variable DIM_J1 : integer := 10;
  variable Seed1 : INTEGER:= 1000;
  variable Seed2 : INTEGER:= 1000;
  variable DIM_I11: REAL;
  variable DIM_J11: REAL;
  variable num: INTEGER:= 30;
  variable VLINE : LINE;
  Seed1 := DIM_J*rel;
  Seed2 := DIM_J*rel;
  num:=DIM_I;
  for i in 1 to num loop
    UNIFORM(Seed1, Seed2, DIM_I11);
    UNIFORM(Seed1, Seed2, DIM_J11);
    DIM_I11:= DIM_I11 * 256.0;
    DIM_J11:= DIM_J11 * 256.0;
    DIM_I1:= INTEGER(DIM_I11);
    DIM_J1:= INTEGER(DIM_J11);
    if (DIM_I1 <= DIM_I and DIM_J1 <= DIM_J) then
      CLUT(DIM_J1, DIM_I1) := INTEGER(rel) + CLUT(DIM_J1, DIM_I1);
      if (CLUT(DIM_J1, DIM_I1) < 0) then
        CLUT(DIM_J1, DIM_I1):= 0;
      elsif (CLUT(DIM_J1, DIM_I1) > 255) then
        CLUT(DIM_J1, DIM_I1):=255;
    end if;
  end loop;
end;
end if;
end if;
Seed1 := INTEGER(DIM_I11*100.0);
Seed2 := INTEGER(DIM_J11*100.0);
end loop;
end;

Procedure SensorGain(num: in integer;
DIM_I, DIM_J: in integer;
CLUT: inout INT_2D(0 to 255, 0 to 255))is

variable I, J, ii, jj, cons: integer:=0;
bEGIN
for J in 0 to DIM_J-1 loop
if ((J >= DIM_J/2 - (DIM_J/8 - 1)) and (J <= DIM_J/2 +(DIM_J/8))) then
jj := 0;
elsif ((J <= DIM_J/2 - (DIM_J/8)) and (J >= DIM_J/2 -(2*DIM_J/8 -1))) or
((J >= DIM_J/2 + (DIM_J/8+1)) and (J <= DIM_J/2 +(2*DIM_J/8))) then
jj := 1;
elsif ((J <= DIM_J/2 - (2*DIM_J/8)) and (J >= DIM_J/2 -(3*DIM_J/8 -1))) or
((J >= DIM_J/2 + (2*DIM_J/8+1)) and (J <= DIM_J/2 +(3*DIM_J/8))) then
jj := 2;
elsif ((J <= DIM_J/2 - (3*DIM_J/8)) and (J >= 0)) or
((J >= DIM_J/2 + (3*DIM_J/8+1)) and (J <= DIM_J-1)) then
jj := 3;
end if;
for I in 0 to DIM_I-1 loop
if ((I >= DIM_I/2 - (DIM_I/8 - 1)) and (I <= DIM_I/2 +(DIM_I/8))) then
ii := 0;
elsif ((I <= DIM_I/2 - (DIM_I/8)) and (I >= DIM_I/2 -(2*DIM_I/8 -1))) or
((I >= DIM_I/2 + (DIM_I/8+1)) and (I <= DIM_I/2 +(2*DIM_I/8))) then
ii := 1;
elsif ((I <= DIM_I/2 - (2*DIM_I/8)) and (I >= DIM_I/2 -(3*DIM_I/8 -1))) or
((I >= DIM_I/2 + (2*DIM_I/8+1)) and (I <= DIM_I/2 +(3*DIM_I/8))) then
ii := 2;
elsif ((I <= DIM_I/2 - (3*DIM_I/8)) and (I >= 0)) or
((I >= DIM_I/2 + (3*DIM_I/8+1)) and (I <= DIM_I-1)) then
ii := 3;
end if;
if (ii <= jj) then
cons := num*jj;
end if;

Appendix A
else
    cons := num*ii;
end if;

CLUT(I, I) := CLUT(J, I) + cons;
    if CLUT(J, I) < 0 then
        CLUT(J, I) := 0;
    elseif (CLUT(J, I) > 255) then
        CLUT(J, I) := 255;
    end if;
end loop;
end loop;
end;

end NoiseFunc;

IRST Mut Prototype

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

entity TARGET_DETECTOR2 is
    port( START_MUT : in BIT;
        INIT       : in BIT;
        IC         : out INTEGER := 0;
        TD         : out INTEGER := 0;
        MIN_SHIFT_X : out INTEGER := 0;
        MIN_SHIFT_Y : out INTEGER := 0;
        TARGET_NUM  : out INTEGER := 0;
        TARGET_X_COORD : out INTEGER := 0;
        TARGET_Y_COORD : out INTEGER := 0;
        TARGET_SPEED : out real := 0.0 );
end TARGET_DETECTOR2;

architecture ALGORITHMIC of TARGET_DETECTOR2 is

constant WH: integer := 64;
constant WL: integer := 64;

Appendix A
constant SCAN_RANGE_X: integer := 3;
constant SCAN_RANGE_Y: integer := 3;
constant THRES_VAL: integer := 4;

Procedure CHEK_FRAME_AVAIL(Ic: in integer;
                              GO : out integer) is

  file  FF_FILE: TEXT is in "frame_no.out";
  variable FIL  : LINE ;
  variable FNO  : integer := 0;
  variable AGAIN : integer := 1;

begin
  while (not(ENDFILE(FF_FILE)) and (FNO<999)) loop
    readline(FF_FILE, FIL);
    read(FIL, FNO);
  end loop;

  if (FNO<Ic) then
    GO:=1;
  else
    GO:=0;
  end if;

end CHEK_FRAME_AVAIL;

FUNCTION Int2Real (II : integer ) return real is

  variable X : real := 0.0;
  variable I : integer := 0;

begin
  I:= abs(II);
  while (I /= 0) loop
    if II > 0 then
      X := X + 1.0;
      I:=I-1;
    end if;
    if II < 0 then
      X := X - 1.0;
      I:=I-1;
  end loop;

  return X;

end Int2Real;
end if;
end loop;
return X;
end Int2Real;

Procedure CHECK_FOR_SHIFT(LI, CL: in ARRAY2D_TYP(1 to WH, 1 to WL);
   DIFF_val: out ARRAY2D_TYP(0 to (2*SCAN_RANGE_X),
                     0 to (2*SCAN_RANGE_Y));
   DIFF_num: out ARRAY2D_TYP(0 to (2*SCAN_RANGE_X),
                     0 to (2*SCAN_RANGE_Y))
) is

variable DIFFt, DIFFn, DIFFx, INDX_X, INDX_Y: integer := 0;

begin

for I in -SCAN_RANGE_Y to SCAN_RANGE_Y loop
   for J in -SCAN_RANGE_X to SCAN_RANGE_X loop
      DIFFt := 0;
      DIFFn := 0;
      DIFFx := 0;
      for Y in 1 to WH loop
         for X in 1 to WL loop
            INDX_X := X + J;
            INDX_Y := Y + I;
            if ((INDX_X >= 1) and (INDX_Y >= 1) and
                (INDX_X <= WL) and (INDX_Y <= WH))
            then
               DIFFt := DIFFt + abs(CL(Y, X) - LI((Y+I),(X+J)));
               DIFFx := DIFFx + 1;
               if (not(abs(CL(Y, X) - LI((Y+I),(X+J)))) = 0) then
                  DIFFn := DIFFn + 1;
               end if;
            end if;
         end loop;
      end loop;
      DIFF_val((I+SCAN_RANGE_Y),(J+SCAN_RANGE_X)) := DIFFt;
      DIFF_num((I+SCAN_RANGE_Y),(J+SCAN_RANGE_X)) := DIFFx;
   end loop;
end loop;
end CHECK_FOR_SHIFT;

Appendix A
Procedure FIND_MIN_SHIFT(DIFF,DIFF_num:in ARRAY2D_TYP(0 to (2*SCAN_RANGE_X),
0 to (2*SCAN_RANGE_Y));
DIFF_MIN: out real;
SHFT_X, SHFT_Y : out integer
)
is

variable I_MAX, J_MAX :integer:=0;
variable DIFF_MIN_t : real:=0.0;

begin

DIFF_MIN_t:=real(DIFF(0,0))/real(DIFF_num(0,0));
SHFT_X:=SCAN_RANGE_X;
SHFT_Y:=SCAN_RANGE_Y;
I_MAX := 2*SCAN_RANGE_Y;
J_MAX := 2*SCAN_RANGE_X;

for I in 0 to I_MAX loop
  for J in 0 to J_MAX loop
    if((real(DIFF(I,J))/real(DIFF_num(I,J))) < DIFF_MIN_t) then
      DIFF_MIN_t := (real(DIFF(I,J))/real(DIFF_num(I,J)));
      SHFT_X := J - SCAN_RANGE_X;
      SHFT_Y := I - SCAN_RANGE_Y;
    end if;
  end loop;
end loop;
DIFF_MIN := DIFF_MIN_t;
end FIND_MIN_SHIFT;

Procedure FIND_DIFF_ARRAY2(DIFF_ARRAY_FILE: out TEXT;
LI,CI:in ARRAY2D_TYP(1 to WH,1 to WL);
ICT, SHFT_X, SHFT_Y : in integer;
tar_x, tar_y: inout integer;
t_present, t_i, t_j: inout integer;
t_frames:inout ARRAY2D_TYP(1 to 16,1 to 16);
DIFF_ARRAY: inout ARRAY3D_TYP( 1 to 5,
  1 to WH, 1 to WL )
)is
variable scan_range_i : integer := 5;
variable scan_range_j : integer := 5;
variable INDX_X, INDX_Y, min_i, min_j, max_i, max_j : integer := 0;
variable DIFF_ARRAY_Y : ARRAY2D_TYP(1 to WH, 1 to WL);
variable DIFF_ARRAY_LINE : LINE;
variable DIFFt, DIFFn, DIFFx, DIFF_max : integer := 0;
variable DIFF_min : real := 0.0;
variable SP : string(1 to 1)
:= " ";
variable m_i, m_j, k, l, j, IDX_i, IDX_j : integer := 0;
variable target, target_mov, frame11, frame22:
ARRAY2D_TYP(1 to 16, 1 to 16) := (others => (others => 0));
variable DIFF_val, DIFF_num, DIFF_xxx :
ARRAY2D_TYP(1 to 2*scan_range_i + 1, 1 to 2*scan_range_i + 1)
:= (others => (others => 0));
variable t_max : integer := 0;
variable i_min, i_max, j_min, j_max : integer := 0;
variable FILE : LINE;

begin
  write(DIFF_ARRAY_LINE, I);$cy
  writeln(DIFF_ARRAY_FILE, DIFF_ARRAY_LINE);
  for Y in 1 to WH loop
    for X in 1 to WL loop
      INDX_X := X + SHFT_X;
      INDX_Y := Y + SHFT_Y;
      if ((INDX_X >= 1) and (INDX_Y >= 1) and
          (INDX_X <= WL) and (INDX_Y <= WH))
        then
          DIFF_ARRAY_Y(Y, X) := abs(CI(Y, X) - LI(INDX_Y, INDX_X));
        else
          DIFF_ARRAY_Y(Y, X) := 256;
        end if;
    write( DIFF_ARRAY_LINE, DIFF_ARRAY_Y(Y, X));
    writeln( DIFF_ARRAY_LINE, SP);
    DIFF_ARRAY_ICt(Y, X) := DIFF_ARRAY_Y(Y, X);
    end loop ;
  writeln( DIFF_ARRAY_FILE, DIFF_ARRAY_LINE);
  end loop ;

  min_i := WH;
  min_j := WL;
  max_i := 0;
max_j := 0;

for i in 1 to WH loop
    for j in 1 to WL loop
        if (DIFF_ARRAY(i,j) > 4 and DIFF_ARRAY(i,j) /= 256) then
            if (min_i > i) then
                min_i := i;
            end if;
            if (max_i < i) then
                max_i := i;
            end if;
            if (min_j > j) then
                min_j := j;
            end if;
            if (max_j < j) then
                max_j := j;
            end if;
        end if;
    end loop;
end loop;

if (max_i - min_i < scan_range_i) then
    scan_range_i := max_i - min_i;
end if;

if (max_j - min_j < scan_range_j) then
    scan_range_j := max_j - min_j;
end if;

if (min_i > max_i or min_j > max_j) then
    t_present := 0;
elseif (max_i-min_i >15 or max_j-max_i > 15) then
    t_present := 2;
end if;

if (min_i<=max_i and min_j<=max_j and max_i-min_i<=15 and max_j-max_i<= 15) then
    t_present := 1;
    m_i:= 0;
    m_j:= 0;
    for i in min_i to max_i loop
        m_i := m_i + 1;
    end loop;
    for j in min_j to max_j loop
        m_j := m_j + 1;
    end loop;
end if;
m_j := 0;
for j in min_j to max_j loop
    m_j := m_j + 1;
    frame11(m_i, m_j) := LI(i+SHFT_Y,j+SHFT_X);
    frame22(m_i, m_j) := CI(i, j);
end loop;
end loop;

tar_x := scan_range_i;
tar_y := scan_range_j;

for i in -scan_range_i to scan_range_i loop
    for j in -scan_range_j to scan_range_j loop
        DIFFt := 0;
        DIFFn := 0;
        DIFFx := 0;
        for k in 1 to m_i loop
            for l in 1 to m_j loop
                IDX_i := i + k;
                IDX_j := j + l;
                if ((IDX_i>=1) and (IDX_j>=1) and
                    (IDX_i<=m_i) and (IDX_j<=m_j)) then
                    DIFFt := DIFFt + abs(frame22(k,l) - frame11(IDX_i,IDX_j));
                    DIFFx := DIFFx + 1;
                    if (not(abs(frame22(k,l) - frame11(IDX_i,IDX_j))=0)) then
                        DIFFn := DIFFn + 1;
                    end if;
                end if;
            end loop;
        end loop;
     end loop;
end loop;

if ((i+scan_range_i+1=1) and (j+scan_range_j+1=1)) then
    DIFF_min := real(DIFFt)/real(DIFFx);
    DIFF_max := DIFFx;
end if;

if ((real(DIFFt)/real(DIFFx) < DIFF_min) or ((real(DIFFt)/real(DIFFx) = DIFF_min)
    and (DIFF_max < DIFFx))) then
    DIFF_min := real(DIFFt)/real(DIFFx);
    DIFF_max := DIFFx;
end if;
target_mov := frame22;

t_max := 0;
for i in 1 to m_i loop
  for j in 1 to m_j loop
    if ((i+tar_y>=1) and (i+tar_y<=m_i))
      and ((j+tar_x>=1) and (j+tar_x<=m_j)) then
      target(i+tar_y,j+tar_x) := frame11(i,j);
    end if;
  end loop;
end loop;
end loop;

for i in 1 to m_i loop
  for j in 1 to m_j loop
    if (target(i,j) = 0) then
      target_mov(i,j) := 0;
    elseif (target(i,j) /= 0) then
      if (abs(target(i,j) - target_mov(i,j)) <= 2) then
        target(i,j) := 0;
      end if;
    end if;
    if (target(i,j) > t_max) then
      t_max := target(i,j);
    end if;
  end loop;
end loop;
if t_max = 0 then
  target := frame22;
end if;

i_min := 1;
i_max := 0;
for i in 1 to m_i loop
  k := 0;
  for j in 1 to m_j loop
    if not(target(i,j) = 0) then
      k := 1;
    end if;
  end loop;
end loop;
end if;
end loop;
if (k = 0) then
  if (i_min = i) then
    i_min := i_min+1;
    i_max := i_max+1;
  end if;
end if;
if (k = 1) then
  i_max := i_max+1;
end if;
end loop;
j_min:=1;
j_max:=0;
for j in 1 to m_j loop
  k := 0;
  for i in 1 to m_i loop
    if not(target(i,j) = 0) then
      k := 1;
    end if;
  end loop;
end if;
if (k = 0) then
  if (j_min = j) then
    j_min := j_min+1;
    j_max := j_max+1;
  end if;
end if;
if (k = 1) then
  j_max := j_max+1;
end if;
end loop;
t_i := 0;
for i in i_min to i_max loop
  t_i := t_i+1;
  t_j := 0;
  for j in j_min to j_max loop
    t_j := t_j+1;
    t_frames(t_i,t_j):= target(i,j);
  end loop;
end loop;
end if;
end FIND_DIFF_ARRAY2;

--------// Architectural Description Begins Here //----------

signal DIFF_S, i_S, J_S : integer := 0;

begin

MAIN:process(START_MUT, INIT)

file INFILE : TEXT is in "frames.txt";
variable FlAG : BIT := '0';
variable SPEED : ARRAY1D_TYP(1 to WH*WL);
variable LI, CI : ARRAY2D_TYP(1 to WH, 1 to WL);
variable DIFF_num, DIFF_val:ARRAY2D_TYP(0 to (2*SCAN_RANGE_X),
0 to (2*SCAN_RANGE_Y));
variable CIL : LINE ;
variable t_p, t_i, t_j, targ_x, targ_y: integer;
variable t_frames: ARRAY2D_TYP(1 to 16, 1 to 16)
:= (others=>(others=>0));
variable GO, STOP, ICC, ICT, I_MAX, J_MAX, SHFT_X,
SHFT_Y, TT_NUM : integer := 0;
variable Diff_MIN:real:=0.0;
variable COORDS: ARRAY2D_REC_TYP(1 to 5, 1 to (WH*WL)) ;
variable DIFF_ARRAY: ARRAY3D_TYP(1 to 5, 1 to WH, 1 to WL)
:=(others =>(others=>(others=>0)));
file DIFF_ARRAY_FILE: TEXT is out " diff.txt";

begin

if (START_MUT'event and START_MUT = '1') then

READLINE(INFILE, CIL);
READ(CIL, ICT);
assert (ICT = (ICC+1))
report "I have lost some frame(s)..."
severity NOTE;

if (ICT /= (ICC+1)) then

ICC:=ICT-1;
end if;
for Ii in 1 to WH loop

READLINE(INFILE, CIL);
for Jj in 1 to WL loop

end loop;
end loop;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end for;
diff Ii in 1 to WH loop

Appendix A

100
READ(CIL, CI(Ii, Jj));
end loop;
end loop;

ICC:=ICC+1;
if ICC > 1 then
   CHECK_FOR_SHIFT(LI, CI, DIFF_val, DIFF_num);
   FIND_MIN_SHIFT(DIFF_val, DIFF_num, DIFF_MIN, SHFT_X, SHFT_Y);
   t_p:= 0;
   t_i:= 0;
   t_j:= 0;
   FIND_DIFF_ARRAY2(DIFF_ARRAY_FILE, LI, CI, ICt,
      SHFT_X, SHFT_Y, targ_x, targ_y, t_p,
      t_i, t_j, t_frames, DIFF_ARRAY);
   TT_NUM := t_p;
   IC <= ICt;
   MIN_SHFT_X <= SHFT_X;
   MIN_SHFT_Y <= SHFT_Y;
   TD <= TT_NUM;
   TARGET_XCOORD <= targ_x;
   TARGET_YCOORD <= targ_y;
end if;
if ( ICC = 1 ) then
   IC <= ICt;
end if;
LI:=CI;
end if;
if (INIT‘event and INIT = ‘0’) then
   STOP:=1;
end if;
end process MAIN;

end ALGORITHMIC;

Infrared Search and Track Structural Model

use work.all;
use STD.TEXTIO.all;
use work.USER_TYPES1.all;
use work.MATH_REAL.all;
entity tb is
end tb;

architecture STRUCTURAL of tb is

signal PER               : TIME;
signal INIT,CLK          : BIT;
signal DIM_I, DIM_J      : INTEGER;
signal NoiTyp, NoiVal, SenGain : INTEGER;
signal START_S, START_T, START_MUT : BIT;
signal SIZE_X_T, SIZE_Y_T, SIZE_X_S, SIZE_Y_S : INTEGER;
signal INIT_X_T, INIT_Y_T, INIT_X_S, INIT_Y_S : INTEGER;
signal DISP_X_T, DISP_Y_T, DISP_X_S, DISP_Y_S : INTEGER;
signal IC,TD,MN_SHT_X,MN_SHT_Y,TARGET_NUM : INTEGER;
signal TARGET_XCOORD, TARGET_YCOORD      : INTEGER;
signal TARGET_SPEED        : REAL;

component OSCA
    Port( PER : TIME;
           INIT : IN BIT;
           CLK : OUT BIT );
end component;

component ENVIRONA
    Port( DIM_I : IN INTEGER;
           DIM_J : IN INTEGER;
           CLK : IN BIT;
           START_T : OUT BIT );
end component;

component TARGETA
    Port( DIM_I : IN INTEGER;
           DIM_J : IN INTEGER;
           SIZE_X : IN INTEGER;
           SIZE_Y : IN INTEGER;
           INIT_X : IN INTEGER;
           INIT_Y : IN INTEGER;
           DISP_X : IN INTEGER;
           DISP_Y : IN INTEGER;
           START_T : IN BIT;
           START_S : OUT BIT );
end component;
component SENSORA
  Port ( DIM_I : IN INTEGER;
        DIM_J : IN INTEGER;
        SIZE_X : IN INTEGER;
        SIZE_Y : IN INTEGER;
        INIT_X : IN INTEGER;
        INIT_Y : IN INTEGER;
        DISP_X : IN INTEGER;
        DISP_Y : IN INTEGER;
        NoiVal : IN INTEGER;
        NoiTyp : IN INTEGER;
        SenGain : IN INTEGER;
        START_S : IN BIT;
        START_MUT : OUT BIT);
end component;

component TARGET_DETECTOR2A
  port ( START_MUT : in BIT;
         INIT : in BIT;
         IC : out INTEGER :=0;
         TD : out INTEGER :=0;
         MIN_SHFT_X : out INTEGER :=0;
         MIN_SHFT_Y : out INTEGER :=0;
         TARGET_NUM : out INTEGER :=0;
         TARGET_X_COORD : out INTEGER :=0;
         TARGET_Y_COORD : out INTEGER :=0;
         TARGET_SPEED : out real := 0.0);
end component;

for G1: OSCA use entity work.OSC(BEHAVIORAL);
for G2: ENVIRONA use entity work.ENVIRON(BEHAVIORAL);
for G3: TARGETA use entity work.TARGET(BEHAVIORAL);
for G4: SENSORA use entity work.SENSOR(BEHAVIORAL);
for G5: TARGET_DETECTOR2A use entity
  work.TARGET_DETECTOR2(ALGORITHMIC);

begin

G1:OSCA
  port map(PER, INIT, CLK);
G2:ENVIRONA
  port map(DIM_I, DIM_J, CLK, START_T);
G3:TARGETA
port map(DIM_I, DIM_J, SIZE_X_T, SIZE_Y_T,
    INIT_X_T, INIT_Y_T, DISP_X_T,
    DISP_Y_T, START_T, START_S);
G4: SENSOR
    port map(DIM_I, DIM_J, SIZE_X_S, SIZE_Y_S,
        INIT_X_S, INIT_Y_S, DISP_X_S,
        DISP_Y_S, NOIVal, NOITyp, SENGain,
        START_S, START_MUT);
G5: TARGET_DETECTOR2A
    port map(START_MUT, INIT, IC, TD,
        MIN_SHIFT_X, MIN_SHIFT_Y, TARGET_NUM,
        TARGET_X_COORD, TARGET_Y_COORD,
        TARGET_SPEED);

end STRUCTURAL;
Appendix B: Template of VHDL Simulation Control File

Simulation control file for ON-LINE Code-Generated Configuration

cd /tb
set tr 1
set COUNT1 999
set ITER 1

open -p OUTPUT_WINDOW
open OUTWIN.txt
open T_Move.txt
open P_Move.txt
open T_Sig.txt

wmove OUTPUT_WINDOW 10,40
wsize OUTPUT_WINDOW 85,40
set X 1
set Y 1
set Y1 5
set Y2 3
set NUM 1

set Frames 4
set DEL 1
set FrTime 250
assign -after 'SDEL ('1') INIT
assign -after 'SFrames*$FrTime+$DEL+$tr ('0') INIT
assign (256) DIM_I
assign (256) DIM_J
assign (250 ms) PER
assign (5) SIZE_X_T
assign (5) SIZE_Y_T
assign (128) INIT_X_T
assign (128) INIT_Y_T
assign (1) DISP_X_T
assign (1) DISP_Y_T
assign (64) SIZE_X_S
assign (64) SIZE_Y_S
assign (96) INIT_X_S
assign (96) INIT_Y_S
assign (-1) DISP_X_S
assign (-1) DISP_Y_S
assign (75) NoiVal
assign (2) NoiTyp
assign (1) SenGain

set RUN_TIME `Frames*4`
set target target.vhd
set sensor sensor.vhd
set osc osc.vhd
monitor -s on target 57
monitor -s on sensor 60
monitor -s on irst_tb_maxi.vhd 459
monitor -s on osc 19

open frame_no.out
while ($ITER <= $Frames) loop
  fprintf "\%d\n" $ITER > frame_no.out
  set ITER `ITER + 1`
end loop
fprintf "\%d\n" $COUNT1 > frame_no.out
close frame_no.out

open gain.out
fprintf "\%d\n" `SenGain` > gain.out
close gain.out

set ITER 4

while ($RUN_TIME >= 0) loop
  run
  set RUN_TIME `RUN_TIME - 1`
  set ITER `ITER - 1`
  if ($ITER = 0) then
    cd G5
  if (main/stop /= 1) then
set ICC 'IC'
echo Frame# $ICC ... > OUTPUT_WINDOW($X,$Y)
fprient "%s %d\n" "Frame# " $ICC > OUTWIN.txt
set X `$X+1`
end if

if (TD=0) then
  if ((MIN_SHFT_X=0) and (MIN_SHFT_Y=0)) then
      echo No Sensor Frame Movement Detected.>OUTPUT_WINDOW($X,$Y)
    fprient "%s\n" "No Sensor Frame Movement Detected.">OUTWIN.txt
  else
    echo The Sensor Frame Shift in Pixels
    (Hz,Vt)=('MIN_SHFT_X', 'MIN_SHFT_Y')
    > OUTPUT_WINDOW($X,$Y)
    fprient "%s%d\n %d%\n%v" "The Sensor Frame
    Shift in Pixels (Hz,Vt) =
    ("MIN_SHFT_X", "MIN_SHFT_Y")"
    > OUTWIN.txt
  end if;
  fprient "%d %d\n" 'MIN_SHFT_X' 'MIN_SHFT_Y' > P_Move.txt
  set X `$X+1`
  echo No Target Detected in this Frame.>OUTPUT_WINDOW($X,$Y)
  fprient "%s\n" "No Target Detected in this Frame."> OUTWIN.txt
  fprient "%d %d\n" 0 0 > T_Sig.txt
  fprient "%d %d\n" 0 0>T_Move.txt
end if
if (TD = 2) then
  echo Sensor Frame Movement Detected!!>OUTPUT_WINDOW($X,$Y)
  set X `$X + 1`
  echo But The Model could not detect the correct
  Sensor Frame Movement. >OUTPUT_WINDOW($X,$Y)
  fprient "%s\n" "Sensor Frame Movement Detected!!"> OUTWIN.txt
  fprient "%s\n" "But The Model could not detect
  the correct Sensor Frame Movement."
  >OUTWIN.txt
  fprient "%d %d\n" 0 0>T_Move.txt
  fprient "%d %d\n" 0 0>P_Move.txt
end if;
if (TD=1) then
  if ((MIN_SHFT_X=0) and (MIN_SHFT_Y=0)) then
      echo No Sensor Frame Movement Detected.>OUTPUT_WINDOW($X,$Y)
    fprient "%s\n" "No Sensor Frame Movement Detected.">OUTWIN.txt
  else

Appendix B
echo "The Sensor Frame Shift in Pixels (Hz,Vt) =
    ('MIN_SHFT_X', -MIN_SHFT_Y') > OUTPUT_WINDOW($X,$Y)
    fprintf "%-s%3d%3s %d%3s\n" "The Sensor Frame Shift in
    Pixels (Hz,Vt) = (" 'MIN_SHFT_X' ", " -MIN_SHFT_Y' ")"
    >OUTWIN.txt
end if;

    fprintf "%d %d\n" "MIN_SHFT_X 'MIN_SHFT_Y' >P_Move.txt
    set X 'X+1'
    echo "Target Detected !! > OUTPUT_WINDOW($X,$Y)
    fprintf "%s\n" "Target Detected !!" >OUTWIN.txt
    set X 'X+1'
    echo "Target Information: > OUTPUT_WINDOW($X,$Y2)
    fprintf "%s\n" "Target Information:" >OUTWIN.txt
    set X 'X+1'
    cd MAIN

    echo "The Target Shift in Pixels (Hz,Vt) =
    ('TARGET_X_COORD', -TARGET_Y_COORD')
    > OUTPUT_WINDOW($X,$Y)
    fprintf "%s%3s %d%3s\n" "The Target Shift in
    Pixels (Hz,Vt) = (" 'TARGET_X_COORD' ",
    " -TARGET_Y_COORD' ")" >OUTWIN.txt
    fprintf "%d %d\n" "TARGET_X_COORD 'TARGET_Y_COORD'

    >T_Move.txt
    set X 'X+1'
    echo "The Probable Target Signature. >OUTPUT_WINDOW($X,$Y)
    fprintf "%s\n" " The Probable Target Signature. "

    >OUTWIN.txt
    set X 'X+1'
    set K 1
    set L 1

    fprintf "%d %d\n" 't_i' 't_j' >T_Sig.txt
    while ($K <= t_i) loop
        while ($L <= t_j) loop
            if (t_frames($K,$L) /= 0) then
                echo "t_frames($K,$L) > OUTPUT_WINDOW($X,$Y)
                fprintf "%-4d\n" 't_frames($K,$L)' >OUTWIN.txt
                fprintf "%-4d\n" 't_frames($K,$L)' > T_Sig.txt
                set Y '$Y+4'
            else
                fprintf "%s\n" " >OUTWIN.txt
                fprintf "%s\n" "0" >T_Sig.txt
                set Y '$Y+4'
            end if
set L '$L+1'
end loop
fprint "\n" >OUTWIN.txt
  fprint "\n" >T_Sig.txt
set X '$X+1'
set Y 1
set K '$K+1'
set L 1
end loop;
set X '$X+1'
  fprint "\n" >OUTWIN.txt
  cd ..
end if
set X '$X+1'
set X '$X+1'
  echo > OUTPUT_WINDOW($X,$Y)
set ITER 4
  cd ..
if ($NUM <$Frames) then
  set NUM `$NUM+1`
else
  set X '$X+1'
  set y 2
  echo All Frames have been Processed! > OUTPUT_WINDOW($X,$Y)
  fprint "%s\n" "All Frames have been Processed!" > OUTWIN.txt
  close OUTWIN.txt
close T_Sig.txt
close P_Move.txt
close T_Move.txt
! /home/rassptbg/DEM0/IRST/CON/Quit
  end if
  end if

end loop

open frames.out
fprint "%d\n" $COUNT1 > frames.out
close frames.out
quit
Appendix C: Math Model of Requirements Interface

call_RequirementsInterface(SPorMP)
int SPorMP;
{
    FILE *fpt1, *fpt2;
    char ch, ch1;
    int temp, tmp[8][8], pix_hh, pix_ll, tar_x, tar_y, pix_h, pix_l, i, j, k, l;
    int DIM_J1, DIM_J11, INIT_Y_T1, INIT_X_T1, PltSpd1X, PltSpd1Y;
    int DISP_X_S, DISP_Y_S;
    int ClutRng1, SenIFOV1, PltSpd1, PltMDir1, TarSpd1, TarTyp1, TarRng1;
    int TarPosNS1, TarPosEW1, TarDirXY1, TarDirZ1, FrRate1;
    int StTime1, NoFrames1;
    int NoiVal1, NoiTyp1, SenGain1, SenSpd1, SIZE_Y_T1, SIZE_X_T1;
    int DISP_X_T1, DISP_Y_T1;
    int INIT_X_S, INIT_Y_S, SIZE_X_S, SIZE_Y_S, EnTime1, PER, INTERVAL;
    float tar[128][128], mx, temp1, TarSpd1XY, TarSpd1X, TarSpd1Y;

    fpt1 = fopen("/tmp/irst.gen", "r");
    fscanf(fpt1, "%d\n%d\n%d\n%d\n", &FrRate1, &StTime1,
        &NoFrames1, &SenIFOV1, &ClutRng1);
    fscanf(fpt1, "%d\n%d\n%d\n%d\n", &TarRng1, &TarPosNS1,
        &TarPosEW1, &TarSpd1, &TarDirXY1);
    fscanf(fpt1, "%d\n%d\n%d\n%d\n", &TarDirZ1, &PltSpd1, &PltMDir1);
    fscanf(fpt1, "%d\n%d\n%d\n%d\n", &SenSpd1, &NoiTyp1, &NoiVal1,
        &SenGain1);

    fclose(fpt1);
    fpt2 = fopen("/tmp/req.out", "w");
    DIM_J11 = DIM_J1 = temp = 0;
    ch1 = 'y';
    if ((fpt1 = fopen("clutter.txt", "r")) == NULL)
    {
        printf("The clutter file does not exist.");
        exit(1);
else
{
    while((ch = fgetc(fpt1)) != EOF)
    {
        switch(ch)
        {
            case '\n': DIM_J1++;  
            DIM_I1 = temp;  
            temp = 0;  
            ch1 = 'y';  
            break;  
            case ' ': if (ch1 == 'y')  
            {  
                ch1 = 'y';  
            }  
            if (ch1 == 'z' || ch1 == 'x')  
            {  
                ch1 = 'x';  
            }  
            break;  
            default: if (ch1 == 'x')  
            {  
                temp++;  
                ch1 = 'z';  
            }  
            if (ch1 == 'z' || ch1 == 'y')  
            {  
                ch1 = 'z';  
            }  
            break;  
        }
    }
}
fclose(fpt1);  
DIM_I1++;  

temp = NoFrames1;  
fprintf(fpt2, "%d\n", temp);  
temp = S:Time1;  
fprintf(fpt2, "%d\n", temp);  
temp = 1000/FrRate1;  
if (FrRate1 == 3)  
    temp++;  
fprintf(fpt2, "%d\n", temp);  

Appendix C
temp = DIM_I1;
fprintf(fp1, "%d\n", temp);
temp = DIM_I1;
fprintf(fp1, "%d\n", temp);
temp = 1000/FrRate1;
fprintf(fp1, "%d\n", temp);

if (SPorMP == 2)
{
    tar_x = tar_y = temp = 0;
    ch1 = 'y';
    if ((fp1 = fopen("target.txt", "r")) == NULL)
    {
        printf("The target file does not exist.");
        exit(1);
    }
    else
    {
        while((ch = fgetc(fp1)) != EOF)
        {
            switch(ch)
            {
            case '\n':
                tar_y++;
                tar_x = temp;
                temp = 0;
                ch1 = 'y';
                break;
            case ' ': if (ch1 == 'y')
            {
                ch1 = 'y';
            }
                if (ch1 == 'z' || ch1 == 'x')
                {
                    ch1 = 'x';
                }
                break;
            default: if (ch1 == 'x')
            {
                temp++;
                ch1 = 'z';
            }
                if (ch1 == 'z' || ch1 == 'y')
                ch1 = 'z';
            }
        }
    }
}
break;
}
}
fclose(fpt1);
tar_x++;

mx = 0.0;
fpt1 = fopen("target.txt", "r");
for(j=0; j<tar_y ; j++)
{
for(i=0;i<tar_x;i++)
{
    fscanf(fpt1,"%f", &tar[j][i]);
    if (tar[j][i]>mx)
        mx = tar[j][i];
}
}
fclose(fpt1);

SIZE_Y_T1 = 200*8/TarRng1;
SIZE_X_T1 = SIZE_Y_T1;

pix_h = tar_y/SIZE_Y_T1;
pix_hh = pix_h/2;
pix_l = tar_x/SIZE_X_T1;
pix_ll = pix_l/2;
fpt1 = fopen("target.txt","w");
for(i = 0;i<SIZE_Y_T1;i++)
{
for(j = 0;j<SIZE_X_T1;j++)
{
    k = i*pix_h+pix_hh-1;
    l = j*pix_l+pix_ll-1;
    tmp[i][j] = nint(255*tar[k][l]/mx);
    if(tmp[i][j] > 255)
        tmp[i][j] = 255;
    if(tmp[i][j] < 0)
        tmp[i][j] = 0;
    fprintf(fpt1,"%4d", tmp[i][j] );
}
fprintf(fpt1,\n"");
}
fclose(fpt1);
temp = SIZE_X_T1;
fprintf(fpt2, "\%d\n", temp);
temp = SIZE_Y_T1;
fprintf(fpt2, "\%d\n", temp);
}
else if(SPorMP == 1)
{
    SIZE_X_T1 = SIZE_Y_T1 = 8;
temp = SIZE_X_T1;
fprintf(fpt2, "\%d\n", temp);
temp = SIZE_Y_T1;
fprintf(fpt2, "\%d\n", temp);
}
/*%%% Finding the Target's initial placement. */

temp = TarRag1*1600/15;
temp1 = ((float)TarPosNS1)/1000;
INIT_Y_T1 = nint(DIM_J1/2 + sin(-temp1)*temp);
temp1 = ((float)TarPosEW1)/1000;
INIT_X_T1 = nint(DIM_J1/2 + sin(temp1)*temp);
temp = INIT_X_T1;
fprintf(fpt2, "\%d\n", temp);
temp = INIT_Y_T1;
fprintf(fpt2, "\%d\n", temp);

/*%%% Finding the Target's Displacement %%%*/

temp1 = (((float)TarDirZ1)*M_PI)/180;
TarSpd1XY = ((float)TarSpd1)*sin(temp1);
temp1 = (((float)TarDirXY1)*M_PI)/180;
TarSpd1X = ((float)TarSpd1XY)*cos(temp1);
TarSpd1Y = ((float)TarSpd1XY)*sin(-(temp1));
DISP_X_T1 = nint((TarSpd1X/FrRate1)/15);
DISP_Y_T1 = nint((TarSpd1Y/FrRate1)/15);
temp = DISP_X_T1;
fprintf(fpt2, "\%d\n", temp);
temp = DISP_Y_T1;
fprintf(fpt2, "\%d\n", temp);

/*%%% Finding the Size of the frame. %%%*/

temp = (96*ClutRng1*SenIFOV1)/(300*3);

Appendix C
SIZE_X_S = SIZE_Y_S = temp;
fpt1 = fopen("/tmp/SensorSizes.txt", "w");
temp = SIZE_X_S;
fprintf(fpt2, "%d\n", temp);
fprintf(fpt1, "%d\n", temp);
temp = SIZE_Y_S;
fprintf(fpt2, "%d\n", temp);
fprintf(fpt1, "%d\n", temp);
fclose(fpt1);

/* Finding The initial Frame placement. */

INIT_X_S = (DIM_I1/2 - SIZE_X_S/2);
INIT_Y_S = (DIM_I1/2 - SIZE_Y_S/2);
temp = INIT_X_S;
fprintf(fpt2, "%d\n", temp);
temp = INIT_Y_S;
fprintf(fpt2, "%d\n", temp);

/* Finding The Frame Displacement, */
/* due to platform motion */

temp1 = ((float)PitMDir1)*M_PI/180;
PltSpd1X = PltSpd1*cos(temp1);
PltSpd1Y = PltSpd1*sin(-(temp1));
temp1 = ((float)PltSpd1X)/((float)(FrRate1*15));
DISP_X_S = nint(temp1);
temp1 = ((float)PltSpd1Y)/((float)(FrRate1*15));
DISP_Y_S = nint(temp1);

/******** due to scanning ***********/
temp = nint(((float)SenSpd1)/FrRate1)*((float)ClutRng1)/9375;
DISP_X_S = DISP_X_S + temp;
temp = DISP_X_S;
fprintf(fpt2, "%d\n", temp);
temp = DISP_Y_S;
fprintf(fpt2, "%d\n", temp);

temp = NoiVal1;
fprintf(fpt2, "%d\n", temp);
temp = NoiTyp1;
fprintf(fpt2, "%d\n", temp);
temp = SenGain1;
fpt1 = fopen("/tmp/SensorGain.txt", "w");
fprintf(fpt1, "%d\n", temp);
fprintf(fpt2, "%d\n", temp);
fclose(fpt1);
fclose(fpt2);
Appendix D: Parser Used for Extraction of System Parameters

purge(string1)
char string1[];
{
    FILE *src,*fp2,*fp3,*fp4,*dest;
    char ch;
    int status, length,status1;
    int i;
    int PARAN;
    int ret;
    char string[80], F_name_1[80], F_name_2[80];
    char name1[30],name2[30];
    char value1[30], value2[30], sysstr[80];
    strcpy(F_name_1,string1);
    strcat(F_name_1,".sch");
    strcpy(sysstr, "sge2vhdl ");
    strcat(sysstr, F_name_1);
    system(sysstr);
    strcpy(F_name_1,string1);
    strcat(F_name_1,".vhd");
    strcpy(F_name_2, sys.parm");
    if((src=fopen(F_name_1,"r"))==NULL)
        {
            printf("Cannot open source file %s!!!", F_name_1);
            exit(1);
        }
    if((fp2=fopen("/tmp/generic.2","w"))==NULL)
        {
            printf("Cannot open intermediate file /tmp/generic.2!!!");
            exit(1);
        }
    while(fscanf(src,"%s",string)==1)
        {

if( !strcmp(string,"Generic"))
  if(fscanf(src,"%s",string)==1)
    if(strcmp(string,"Map"))
      continue;
  else
    {
      PARAN=0;
      status=0;
      while (PARAN==0 && (ch=fgetc(src))!=EOF)
        switch (ch)
          { case ':': fputc(’,’,fp2);
               break;
              case ’ ’: break;
              case ’:’: do { ch=fgetc(src);
                } while (ch!=’=’);
              fputc(’=’,fp2);
              fputc(’>’,fp2);
              break;
              case ’)’: PARAN=1;
              break;
              default: fputc(ch,fp2);
            }
      }
  }
}
fputc(’,’,fp2);
fflush(fp2);
fclose(fp2);
fclose(src);
if((fp2=fopen("/tmp/generic.2","r"))==NULL)
  { printf("Cannot open intermediate file /tmp/generic.2!!!");
    exit(1);
  }
if((fp3=fopen("/tmp/generic.3","w"))==NULL)
  { printf("Cannot open intermediate file /tmp/generic.3!!!");
    exit(1);
  }
status1=status=0;
length=0;
while ((ch=fgetc(fp2))!=EOF)
  { switch (ch)
     { case ’=’: status=1;
       break;
case '>', if (status==1) status=2;
    else status = 0;
    break;
    case 'n':
    case ':(':
    case ':,': length=0;
    break;
    default: status=0;
        length++;
    }
    if (status==2)
    { fseek(fp2, (long) -(length+2L), 1L);
    for (i=0;i<length;i++)
        { ch=fgetc(fp2);
          fputc(ch,fp3);
        }
    fgetc(fp2);
    fgetc(fp2);
    fputc(' ',fp3);
    ch=fgetc(fp2);
    while(ch!=')' && ch!=',')
    { fputc(ch,fp3);
      ch=fgetc(fp2);
    }
    fputc('n',fp3);
    length=0;
    status=0;
    }
    fflush(fp3);
    fclose(fp3);
    fclose(fp2);
    system("sort /tmp/generic.3 > /tmp/generic.4");
    if((fp4=fopen("/tmp/generic.4","r"))==NULL)
    { printf("Cannot open intermediate file /tmp/generic.4!!!");
      exit(1);
    }
    if((dest=fopen(F_name_2,"w"))==NULL)
    { printf("Cannot open destination file %s!!!",F_name_2);
      exit(1);
    }
    strcpy(name1,"");
    strcpy(value1,"");
ret=fscanf(fp4,"%s\%s",name2,value2);
while (ret==2)
{
    if (strcmp(name1,name2)!=0)
        fprintf(dest,"%s\n",value2);
    strcpy(name1,name2);
    strcpy(value1,value2);
    ret=fscanf(fp4,"%s\%s",name2,value2);
}
fflush(dest);
fclose(dest);
fclose(fp4);
system("rm -f /tmp/generic.*");
Appendix E: Code Listing For Test Plan Interface

C Function For Requirements Mapping Of The Test Plan Interface

/*
* Function to prompt the user for input values for the set of requirements corresponding
* to the test group selected and to specify the values of remaining parameters
*/
void
IRSTCdgRequireMap(MANorAUTO, TP_num)
int MANorAUTO, TP_num;
{
    if (MANorAUTO == 1)
    {
        xv_set(IRST_IRSTCdgManualInput->IRSTCdgManualInputSaveButton,
                XV_SHOW, TRUE, NULL);
        xv_set(IRST_IRSTCdgManualInput->IRSTCdgManualInputLoadButton,
                XV_SHOW, TRUE, NULL);
    }
    else if (MANorAUTO == 2)
    {
        xv_set(IRST_IRSTCdgManualInput->IRSTCdgManualInputSaveButton,
                XV_SHOW, FALSE, NULL);
        xv_set(IRST_IRSTCdgManualInput->IRSTCdgManualInputLoadButton,
                XV_SHOW, FALSE, NULL);
    }
    /*ClutRng = 300;*/
    ClutRng = SYS_ClutRng;
    if (TP_num == 0 || TP_num == 5 || TP_num == 1 || TP_num == 2 || TP_num == 4)
    {
        xv_set(IRST_IRSTCdgManualInput->
                IRSTCdgManualInputClutRngSlider,
                PANEL_VALUE, ClutRng, PANEL_INACTIVE, FALSE,
                PANEL_MIN_VALUE, 150, PANEL_MAX_VALUE, 300,
                NULL);
    }
    else if (TP_num == 3)
    {

xv_set(IRST_IRSTCdManualInput->
    IRSTCdManualInputClutRngSlider,
    PANEL_VALUE, ClutRng,
    PANEL_INACTIVE, TRUE, NULL);
}

if (MANorAUTO == 2 && TP_num == 2)
{
    PltTyp = 1;
    PltSpd = 10;
    PltMDir = SYS_PltMDir;
    xv_set(IRST_IRSTCdManualInput->
        IRSTCdManualInputPltTypSettings,
        PANEL_VALUE, PltTyp,
        PANEL_CHOOSE_ONE, TRUE, PANEL_CHOOSE_NONE,
        FALSE, PANEL_INACTIVE, TRUE, NULL);
    xv_set(IRST_IRSTCdManualInput->IRSTCdManualInputPltSpdSlider,
        PANEL_VALUE, PltSpd, PANEL_INACTIVE, FALSE,
        PANEL_MIN_VALUE, 0, PANEL_MAX_VALUE, 50,
        PANEL_SHOW_RANGE, TRUE,
        PANEL_SLIDER_END_BOXES, TRUE, NULL);
    xv_set(IRST_IRSTCdManualInput->
        IRSTCdManualInputPltMDirSlider,
        PANEL_VALUE, PltMDir, PANEL_INACTIVE, FALSE,
        PANEL_MIN_VALUE, 0, PANEL_MAX_VALUE, 360,
        PANEL_SHOW_RANGE, TRUE,
        PANEL_SLIDER_END_BOXES, TRUE, NULL);
    xv_set(IRST_IRSTCdManualInput->IRSTCdManualInputPltSpdSlider,
        PANEL_LABEL_STRING,
        "Platform Speed Increment (in M/S) :", NULL);
}
else
{
    PltTyp = 999;
    xv_set(IRST_IRSTCdManualInput->IRSTCdManualInputPltSpdSlider,
        PANEL_LABEL_STRING,
        "Platform Speed (in M/S) :", NULL);
    if(TP_num==0||TP_num == 5||TP_num == 2||TP_num == 3||TP_num == 4)
    {
        xv_set(IRST_IRSTCdManualInput->
            IRSTCdManualInputPltTypSettings,
            PANEL_CHOOSE_ONE, TRUE,
            PANEL_CHOOSE_NONE, TRUE,
PANEL_INACTIVE, FALSE, NULL);
}
else if(TP_num == 1)
{
    xv_set(IRST_IRSTCdgManualInput->
            IRSTCdgManualInputPltTypSettings,
            PANEL_CHOOSE_ONE, TRUE,
            PANEL_CHOOSE_NONE, TRUE,
            PANEL_INACTIVE, TRUE, NULL);
}
PltSpd = 0;
    xv_set(IRST_IRSTCdgManualInput->IRSTCdgManualInputPltSpdSlider,
            PANEL_VALUE, PltSpd, PANEL_INACTIVE, TRUE,
            PANEL_SHOW_RANGE, FALSE,
            PANEL_SLIDER_END_BOXES, FALSE, NULL);
PltMDir = SYS_PltMDir;
    xv_set(IRST_IRSTCdgManualInput->
            IRSTCdgManualInputPltMDirSlider,
            PANEL_VALUE, PltMDir, PANEL_INACTIVE, TRUE,
            PANEL_SHOW_RANGE, FALSE,
            PANEL_SLIDER_END_BOXES, FALSE, NULL);
SenSpd = 0;
    xv_set(IRST_IRSTCdgManualInput->IRSTCdgManualInputSenSpdSlider,
            PANEL_VALUE, SenSpd, PANEL_INACTIVE, TRUE,
            PANEL_SHOW_RANGE, FALSE,
            PANEL_SLIDER_END_BOXES, FALSE, NULL);
NoiVal = SYS_NoIVal;
    xv_set(IRST_IRSTCdgManualInput->IRSTCdgManualInputNoiValSlider,
            PANEL_VALUE, NoiVal, PANEL_INACTIVE, TRUE,
            PANEL_SHOW_RANGE, FALSE,
            PANEL_SLIDER_END_BOXES, FALSE, NULL);
if (MANorAUTO == 2 && TP_num == 1)
{
    TarSpd = 10;
    xv_set(IRST_IRSTCdgManualInput->IRSTCdgManualInputTarSpdSlider,
            PANEL_VALUE, TarSpd, PANEL_INACTIVE, FALSE,
            PANEL_MIN_VALUE, 0, PANEL_MAX_VALUE, 50, NULL);
    xv_set(IRST_IRSTCdgManualInput->IRSTCdgManualInputTarSpdSlider,
            PANEL_LABEL_STRING, "Target Speed Increment (in M/S)",
            NULL);
}
else
{
TarSpd = SYS_TarSpd;
XV_set(IRST_IRSTCdgManualInput->IRSTCdgManualInputTarSpdSlider,
       PANEL_LABEL_STRING, "Target Speed (in M/S) :",
       NULL);
if (TP_num == 0 || TP_num == 5 || TP_num == 1 || TP_num == 4)
{
XV_set(IRST_IRSTCdgManualInput>
       IRSTCdgManualInputTarSpdSlider,
       PANEL_VALUE, TarSpd, PANEL_INACTIVE, FALSE,
       PANEL_MIN_VALUE, 50, PANEL_MAX_VALUE, 500,
       NULL);
}
else if (TP_num == 2 || TP_num == 3)
{
XV_set(IRST_IRSTCdgManualInput>
       IRSTCdgManualInputTarSpdSlider,
       PANEL_VALUE, TarSpd, PANEL_INACTIVE, TRUE,
       PANEL_MIN_VALUE, 50, PANEL_MAX_VALUE, 500,
       NULL);
}
}
TarRng = SYS_TarRng;
if (TP_num == 0 || TP_num == 5 || TP_num == 1 || TP_num == 4)
{
XV_set(IRST_IRSTCdgManualInput->IRSTCdgManualInputTarRngSlider,
       PANEL_VALUE, TarRng, PANEL_INACTIVE, FALSE,
       PANEL_MIN_VALUE, 200, PANEL_MAX_VALUE, 400,
       NULL);
}
else if (TP_num == 2 || TP_num == 3)
{
XV_set(IRST_IRSTCdgManualInput->IRSTCdgManualInputTarRngSlider,
       PANEL_VALUE, TarRng, PANEL_INACTIVE, TRUE,
       PANEL_MIN_VALUE, 200, PANEL_MAX_VALUE, 400,
       NULL);
}
TarPosNS = SYS_TarPosNS;
TarPosEW = SYS_TarPosEW;
TarDirXY = SYS_TarDirXY;
TarDirZ = SYS_TarDirZ;
SenIFOV = SYS_SenIFOV;

Appendix E
NoFrames = SYS_NoFrames;
StTime = 1;
if (TP_num == 0 || TP_num == 5)
{
    xv_set(IRST_IRSTCdgManualInput-
        IRSTCdgManualInputTarPosNSSlider,
        PANEL_VALUE, TarPosNS, PANEL_INACTIVE, FALSE,
        PANEL_MIN_VALUE, -3, PANEL_MAX_VALUE, 3, NULL);
    xv_set(IRST_IRSTCdgManualInput-
        IRSTCdgManualInputTarPosEWSlider,
        PANEL_VALUE, TarPosEW, PANEL_INACTIVE, FALSE,
        PANEL_MIN_VALUE, -3, PANEL_MAX_VALUE, 3, NULL);
    xv_set(IRST_IRSTCdgManualInput-
        IRSTCdgManualInputTarDirXYSlider,
        PANEL_VALUE, TarDirXY, PANEL_INACTIVE, FALSE,
        PANEL_MIN_VALUE, 0, PANEL_MAX_VALUE, 360, NULL);
    xv_set(IRST_IRSTCdgManualInput-
        IRSTCdgManualInputTarDirZSlider,
        PANEL_VALUE, TarDirZ, PANEL_INACTIVE, FALSE,
        PANEL_MIN_VALUE, 0, PANEL_MAX_VALUE, 180, NULL);
    xv_set(IRST_IRSTCdgManualInput->IRSTCdgManualInputFOVSlider,
        PANEL_VALUE, ScnFOV,
        PANEL_INACTIVE, FALSE, PANEL_MIN_VALUE, 2,
        PANEL_MAX_VALUE, 4, NULL);
    xv_set(IRST_IRSTCdgManualInput-
        IRSTCdgManualInputNoFramesSlider,
        PANEL_VALUE, NoFrames, PANEL_INACTIVE, FALSE,
        PANEL_MIN_VALUE, 2, PANEL_MAX_VALUE, 10, NULL);
    xv_set(IRST_IRSTCdgManualInput->IRSTCdgManualInputStTimeSlider,
        PANEL_VALUE, StTime, XV_SHOW, FALSE,
        PANEL_INACTIVE, FALSE, PANEL_MIN_VALUE, 0,
        PANEL_MAX_VALUE, 4, NULL);
}
else if(TP_num == 1 || TP_num == 2 || TP_num == 3 || TP_num == 4)
{
    xv_set(IRST_IRSTCdgManualInput-
        IRSTCdgManualInputTarPosNSSlider, PANEL_VALUE,
        TarPosNS, PANEL_INACTIVE, TRUE,
        PANEL_MIN_VALUE, -3, PANEL_MAX_VALUE, 3, NULL);
    xv_set(IRST_IRSTCdgManualInput-
        IRSTCdgManualInputTarPosEWSlider, PANEL_VALUE,
        TarPosEW, PANEL_INACTIVE, TRUE,
        PANEL_MIN_VALUE, -3, PANEL_MAX_VALUE, 3,
xv_set(IRSTIRSTCdgManualInput->
   IRSTCdgManualInputTarDirXYSlider, PANEL_VALUE, TarDirXY, PANEL_INACTIVE, TRUE,
   PANEL_MIN_VALUE, 0, PANEL_MAX_VALUE, 360, NULL);
xv_set(IRSTIRSTCdgManualInput->
   IRSTCdgManualInputTarDirZSlider, PANEL_VALUE, TarDirZ, PANEL_INACTIVE, TRUE,
   PANEL_MIN_VALUE, 0, PANEL_MAX_VALUE, 180, NULL);
xv_set(IRSTIRSTCdgManualInput->IRSTCdgManualInputFOVSlider,
   PANEL_VALUE, SenIFOV, PANEL_INACTIVE, TRUE,
   PANEL_MIN_VALUE, 2, PANEL_MAX_VALUE, 4, NULL);
xv_set(IRSTIRSTCdgManualInput->
   IRSTCdgManualInputNoFramesSlider, PANEL_VALUE, NoFrames, PANEL_INACTIVE, TRUE,
   PANEL_MIN_VALUE, 2, PANEL_MAX_VALUE, 10, NULL);
xv_set(IRSTIRSTCdgManualInput->IRSTCdgManualInputSTimeSlider,
   PANEL_VALUE, StTime, XV_SHOW, FALSE,
   PANEL_INACTIVE, TRUE,
   PANEL_MIN_VALUE, 0, PANEL_MAX_VALUE, 4, NULL);
}
FrRate = SYS_FrRate;
if (TP_num == 0 || TP_num == 5 || TP_num == 1 || TP_num == 2)
{
    xv_set(IRSTIRSTCdgManualInput->
       IRSTCdgManualInputFrProTimSlider, PANEL_VALUE, FrRate,
       PANEL_INACTIVE, FALSE,
       PANEL_MIN_VALUE, 2, PANEL_MAX_VALUE, 5, NULL);
} else if (TP_num == 3 || TP_num == 4)
{
    xv_set(IRSTIRSTCdgManualInput->
       IRSTCdgManualInputFrProTimSlider, PANEL_VALUE, FrRate,
       PANEL_INACTIVE, TRUE,
       PANEL_MIN_VALUE, 2, PANEL_MAX_VALUE, 5, NULL);
} else
{
    xv_set(IRSTIRSTCdgManualInput->
       IRSTCdgManualInputSenTypSettings, PANEL_INACTIVE, TRUE,
       PANEL_CHOICE_NONE, TRUE, NULL);
}
else if(TP_num == 1 || TP_num == 3)
{
    xv_set(IRST_IRSTCdgManualInput->
        IRSTCdgManualInputSenTypSettings, PANEL_INACTIVE,
        TRUE, PANEL_CHOICE_ONE, TRUE,
        PANEL_CHOICE_NONE, TRUE, NULL);
}
{
    SenGain = SYS_SenGain;
    NoiTyp = 999;
    NoiVal = 0;
    if (TP_num == 0 || TP_num == 5 || TP_num == 3)
    {
        xv_set(IRST_IRSTCdgManualInput->
            IRSTCdgManualInputSenGainSlider, PANEL_VALUE,
            SenGain, PANEL_INACTIVE, FALSE,
            PANEL_MIN_VALUE, -4, PANEL_MAX_VALUE, 4,
            NULL);
        xv_set(IRST_IRSTCdgManualInput->
            IRSTCdgManualInputNoiTypSettings,
            PANEL_INACTIVE, FALSE, PANEL_CHOICE_ONE,
            TRUE, PANEL_CHOICE_NONE, TRUE, NULL);
    }
else if(TP_num == 1 || TP_num == 2 || TP_num == 4)
{
    xv_set(IRST_IRSTCdgManualInput->
        IRSTCdgManualInputSenGainSlider, PANEL_VALUE, SenGain,
        PANEL_INACTIVE, TRUE,
        PANEL_MIN_VALUE, -4, PANEL_MAX_VALUE, 4,
        NULL);
    xv_set(IRST_IRSTCdgManualInput->
        IRSTCdgManualInputNoiTypSettings,
        PANEL_INACTIVE, TRUE, PANEL_CHOICE_ONE,
        TRUE, PANEL_CHOICE_NONE, TRUE, NULL);
}
}
/ * Function to store the primary parameters to be read by the requirements interface */
void
call_TPGenInterface(MANorAUTO, TP_num)
int MANorAUTO, TP_num;
{
    if (MANorAUTO == 2)
    {
        int FrameRate, SimStTime, NumberFrames, FOView, ClutRange;
        int TarRange;
        int TargetPosNS, TargetPosEW, TarSpeed, TargetDirXY, TargetDirZ;
        int PltSpeed, PltMDirection, SenSpeed, NioType, NoiValue, SensorGain;
        FILE *fp;
        fp = fopen("/tmp/irst.gen", "r");
        fscanf(fp, "%d\n%d\n%d\n%d\n\n", &FrameRate, &SimStTime,
               &NumberFrames, &FOView, &ClutRange, &TarRange);
        fscanf(fp, "%d\n%d\n%d\n%d\n\n", &TargetPosNS, &TargetPosEW,
               &TarSpeed, &TargetDirXY, &TargetDirZ);
        fscanf(fp, "%d\n%d\n%d\n%d\n\n", &PltSpeed, &PltMDirection,
               &SenSpeed, &NioType, &NoiValue, &SensorGain);
        fclose(fp);
        if (TP_num == 1)
        {
            fp = fopen("/tmp/irst.gen", "w");
            fprintf(fp, "%d\n%d\n%d\n%d\n%d\n\n", FrameRate,
                     SimStTime, NumberFrames, FOView, ClutRange,
                     TarRange);
            fprintf(fp, "%d\n%d\n%d\n%d\n%d\n\n", TargetPosNS,
                     TargetPosEW, DefaultTarSpd, TargetDirXY, TargetDirZ);
            fprintf(fp, "%d\n%d\n%d\n%d\n%d\n\n", PltSpeed,
                     PltMDirection,SenSpeed, NioType, NoiValue, SensorGain);
            fclose(fp);
        }
        else if (TP_num == 2)
        {
            fp = fopen("/tmp/irst.gen", "w");
            fprintf(fp, "%d\n%d\n%d\n%d\n%d\n\n", FrameRate,
                     SimStTime, NumberFrames, FOView, ClutRange,
                     TarRange);
            fprintf(fp, "%d\n%d\n%d\n%d\n%d\n\n", TargetPosNS,
                     TargetPosEW, TarSpeed, TargetDirXY, TargetDirZ);
        }
    }
}
fprintf(fp, "%d\n%d\n%d\n%d\n%d\n%d\n%d\n", DefaultPltSpd, PltMDirection, SenSpeed, NioType, NoiValue, SensorGain);
fclose(fp);
}
else if (TP_num == 3)
{
    fp = fopen("/tmp/first.gen", "w");
    fprintf(fp, "%d\n%d\n%d\n%d\n%d\n", FrameRate, SimStTime, NumberFrames, FOView, ClutRange, TarRange);
    fprintf(fp, "%d\n%d\n%d\n%d\n", TargetPosNS, TargetPosEW, TarSpeed, TargetDirXY, TargetDirZ);
    fprintf(fp, "%d\n%d\n%d\n%d\n", PltSpeed, PltMDirection, SenSpeed, NioType, DefaultNoiVal, SensorGain);
    fclose(fp);
}
else if (TP_num == 4)
{
    char sysstr[160];
    strcpy(sysstr, "cp ");
    strcat(sysstr, CLUT_dir);
    strcat(sysstr, DefaultClutterNam);
    strcat(sysstr, " ");
    strcat(sysstr, simfiles_dir);
    strcat(sysstr, clut_file);
    system(sysstr);
}

Functions To Generate the VHDL Simulation and Control File

/*
 * Function to Generate the signal assignment statements of the VHDL Simulation and
 * Control File with the values of secondary parameters from the requirements interface
 */
call_ConFileTBPartGenerator()

 Appendix E
static char line1_1[25] = "set Frames ";
static char line2_1[25] = "set DEL ";
static char line3_1[25] = "set FrTime ";
static char line4_1[25] = "assign (1') INIT";
static char line4_2[35] = "assign -after ‘$DEL‘ (‘1’) INIT";
static char line5[55] = "assign -after ‘$Frames*$FrTime+$DEL+${tr} (‘0’) INIT";
static char linegen[10] = "assign (’;
static char line6_2[10] = "”) DiM_I";
static char line7_2[10] = "”) DIM_J";
static char line8_2[10] = "”) ms) PER";
static char line9_2[15] = "”) SIZE_X_T";
static char line10_2[15] = "”) SIZE_Y_T";
static char line11_2[15] = "”) INIT_X_T";
static char line12_2[15] = "”) INIT_Y_T";
static char line13_2[15] = "”) DISP_X_T";
static char line14_2[15] = "”) DISP_Y_T";
static char line15_2[15] = "”) SIZE_X_S";
static char line16_2[15] = "”) SIZE_Y_S";
static char line17_2[15] = "”) INIT_X_S";
static char line18_2[15] = "”) INIT_Y_S";
static char line19_2[15] = "”) DISP_X_S";
static char line20_2[15] = "”) DISP_Y_S";
static char line21_1[15] = "”) NoiVal";
static char line22_2[15] = "”) NoiTyp";
static char line23_2[15] = "”) SenGain";
FILE *ftp1, *ftp2;
char *gen;
int i = 0;
char *line1, *line2, *line3, *line4, *line6;
char *line7, *line8, *line9, *line10;
char *line11, *line12, *line13, *line14, *line15;
char *line16, *line17, *line18, *line19, *line20;
char *line21, *line22, *line23;

ftp1 = fopen("/tmp/req.out", "r");
ftp2 = fopen("/home/rassptbg/DEMO/IRST/CON/con_gen/test.con", "w");

gen = (char *) calloc(1,50 * sizeof(char));
line1 = (char *) calloc(1,50 * sizeof(char));
line2 = (char *) calloc(1,50 * sizeof(char));
line3 = (char *) calloc(1,50 * sizeof(char));
line4 = (char *) calloc(1,50 * sizeof(char));
line5 = (char *) calloc(1,50 * sizeof(char));
line6 = (char *) calloc(1,50 * sizeof(char));
line7 = (char *) calloc(1,50 * sizeof(char));
line8 = (char *) calloc(1,50 * sizeof(char));
line9 = (char *) calloc(1,50 * sizeof(char));
line10 = (char *) calloc(1,50 * sizeof(char));
line11 = (char *) calloc(1,50 * sizeof(char));
line12 = (char *) calloc(1,50 * sizeof(char));
line13 = (char *) calloc(1,50 * sizeof(char));
line14 = (char *) calloc(1,50 * sizeof(char));
line15 = (char *) calloc(1,50 * sizeof(char));

Appendix E
line17=(char *) calloc(1,50 * sizeof(char)); line18=(char *) calloc(1,50 * sizeof(char));
line19=(char *) calloc(1,50 * sizeof(char)); line20=(char *) calloc(1,50 * sizeof(char));
line21=(char *) calloc(1,50 * sizeof(char)); line22=(char *) calloc(1,50 * sizeof(char));
line23 = (char *) calloc(1,50 * sizeof(char));
strcpy(line1, line1_1); strcpy(line2, line2_1); strcpy(line3, line3_1);
strcpy(line6, linegen); strcpy(line7, linegen); strcpy(line8, linegen);
strcpy(line9, linegen); strcpy(line10, linegen); strcpy(line11, linegen);
strcpy(line12, linegen); strcpy(line13, linegen); strcpy(line14, linegen);
strcpy(line15, linegen); strcpy(line16, linegen); strcpy(line17, linegen);
strcpy(line18, linegen); strcpy(line19, linegen); strcpy(line20, linegen);
strcpy(line21, linegen); strcpy(line22, linegen); strcpy(line23, linegen);
 fscanf(fpt1, "%s
", gen);
strcat(line1, gen);
 fscanf(fpt1, "%s
", gen);
strcat(line2, gen);
if (atoi(gen) == 0)
{
    i = 1;
}
 fscanf(fpt1, "%s
", gen); strcat(line3, gen);
 fscanf(fpt1, "%s
", gen); strcat(line6, gen); strcat(line6_2);
 fscanf(fpt1, "%s
", gen); strcat(line7, gen); strcat(line7_2);
 fscanf(fpt1, "%s
", gen); strcat(line8, gen); strcat(line8_2);
 fscanf(fpt1, "%s
", gen); strcat(line9, gen); strcat(line9_2);
 fscanf(fpt1, "%s
", gen); strcat(line10, gen); strcat(line10_2);
 fscanf(fpt1, "%s
", gen); strcat(line11, gen); strcat(line11_2);
 fscanf(fpt1, "%s
", gen); strcat(line12, gen); strcat(line12_2);
 fscanf(fpt1, "%s
", gen); strcat(line13, gen); strcat(line13_2);
 fscanf(fpt1, "%s
", gen); strcat(line14, gen); strcat(line14_2);
 fscanf(fpt1, "%s
", gen); strcat(line15, gen); strcat(line15_2);
 fscanf(fpt1, "%s
", gen); strcat(line16, gen); strcat(line16_2);
 fscanf(fpt1, "%s
", gen); strcat(line17, gen); strcat(line17_2);
 fscanf(fpt1, "%s
", gen); strcat(line18, gen); strcat(line18_2);
 fscanf(fpt1, "%s
", gen); strcat(line19, gen); strcat(line19_2);
 fscanf(fpt1, "%s
", gen); strcat(line20, gen); strcat(line20_2);
 fscanf(fpt1, "%s
", gen); strcat(line21, gen); strcat(line21_2);
 fscanf(fpt1, "%s
", gen); strcat(line22, gen); strcat(line22_2);
 fscanf(fpt1, "%s
", gen); strcat(line23, gen); strcat(line23_2);
fclose(fpt1);

if (i == 1)
{
    fprintf(fpt2, "\n%\n%\n%\n%\n%\n", line1, line2, line3, line4_1, line5);

Appendix E
} else {
    fprintf(fp2, "%s\n%s\n%s\n%s\n", line1, line2, line3, line4_2, line5);
}
fprintf(fp2, "%s\n%s\n%s\n%s\n", line6, line7, line8, line9, line10);
fprintf(fp2, "%s\n%s\n%s\n%s\n", line11, line12, line13, line14, line15);
fprintf(fp2, "%s\n%s\n%s\n%s\n", line16, line17, line18, line19, line20);
fprintf(fp2, "%s\n%s\n%s\n", line21, line22, line23);
fclose(fp2);

/*
 * Function to form the complete VHDL Simulation and Control File By merging the
 * signal assignment statements with templates of VHDL control file
 */
void call_ConFileGenerator(MANorAUTO, TP_num, TBorTBMUT, ONorPOST)
int MANorAUTO, TP_num, TBorTBMUT, ONorPOST;
{
    char CurrentDir[80];
    if ((MANorAUTO == 1) || ((MANorAUTO == 2) && (TP_num == 5)))
    {
        if (TBorTBMUT == 1)
        {
            /* copy the model_alone.con*/
            getwd(CurrentDir);
            chdir(con_gen_dir);
            call_ConFileTBPartGenerator();
            system("rm -f
/home/rassptbg/DEMO/IRST/CON/model_alone.con");
            system("cat model_alone.head test.con model_alone.tail >
/home/rassptbg/DEMO/IRST/CON/model_alone.con");
            chdir(CurrentDir);
        }
        if (TBorTBMUT == 2)
        {
            if (ONorPOST == 1)
            {
                /* copy the model_tb_mutex_online.con*/
                getwd(CurrentDir);
                chdir(con_gen_dir);
                call_ConFileTBPartGenerator();
            }
        }
    }

Appendix E
system("rm -f /home/rassptbg/DEMO/IRST/CON/model_tb_mut_online.con");
    system("cat model_tb_mut_online.head test.con model_tb_mut_online.tail >
/home/rassptbg/DEMO/IRST/CON/model_tb_mut_online.con");
    chdir(CurrentDir);
}
if (ONorPOST == 2)
{
    /* copy the model_alone.con*/
    /* copy irst_alone_online.con*/
    getwd(CurrentDir);
    chdir(con_gen_dir);
    call_ConFileTBPartGenerator();
    system("rm -f /home/rassptbg/DEMO/IRST/CON/model_alone.con");
    system("cat model_alone.head test.con model_alone.tail >
/home/rassptbg/DEMO/IRST/CON/model_alone.con");
    chdir(CurrentDir);
}
}
if ((MANorAUTO == 2) && ((TP_num == 1)||(TP_num == 2)||(TP_num == 3)||(TP_num == 4)))
{
    if (TBo:TBMUT == 2)
    {
        if (ONorPOST == 1)
        {
            /* copy model_tb_mut_post.con*/
            getwd(CurrentDir);
            chdir(con_gen_dir);
            call_ConFileTBPartGenerator();
            system("rm -f /home/rassptbg/DEMO/IRST/CON/model_tb_mut_post.con");
            system("cat model_tb_mut_post.head test.con model_tb_mut_post.tail >
/home/rassptbg/DEMO/IRST/CON/model_tb_mut_post.con");
            chdir(CurrentDir);
        }
        if (ONorPOST == 2)
        {
            /* copy the model_alone.con*/
            /* copy irst_alone_post.con*/

Appendix E 133
getwd(CurrentDir);
chdir(con_gen_dir);
call_ConFileTBPartGenerator();
system("rm -f 
/home/rassptbg/DEMO/IRST/CON/model_alone.con");
system("cat model_alone.head test.con model_alone.tail > 
/home/rassptbg/DEMO/IRST/CON/model_alone.con");
chdir(CurrentDir);
Vita

Sailesh Kottapalli was born on the 4th of December, 1972 in Gannavaram, India. He graduated with a Bachelors of Technology degree in Computer Science and Systems Engineering from the College of Engineering, Andhra University, in June 1994. He attended graduate school at Virginia Polytechnic Institute and State University and received a Master of Science degree in Electrical Engineering in May 1996. He has been employed with Intel Corporation, Santa Clara since May 1996.