High Level Generation of VHDL Test Benches

James R. Armstrong, EE Department, Virginia Tech
Geoffrey Frank, Research Triangle Institute
Srinivasan Hrishikesh
Prabhakar Gowrisankaran
Zhen Xu
EE Department, Virginia Tech

Abstract

The generation of testbenches for complicated VHDL models is a labor intensive task rivaling that of generating the model itself. What is required is a high level approach to test bench development which relieves the modeler of the details of this process. In this paper, methods employing CASE tools, design tools and model libraries are described, which allow the modeler to develop testbenches at a high level of abstraction and thus reduce the amount of labor required for testbench development.

1: Introduction

In the approach described in this paper, testbench development is driven directly and in an automated fashion by specification requirements which describe the system (which the model represents) and the environment in which the system resides. CASE and high level design tools are used in the test bench generation process. The methodology has been successfully applied to generate testbenches for models of DSP algorithms on the RASSP program.

2: Basic Approach

Test generation for models can be model based or environment based. In the model based approach, the model itself is used to develop tests. Conventional gate level ATPG [1] is an example of model based test generation. Approaches to model based test generation at the behavioral level are also being investigated [2,3]. In environment based testing one develops tests from the environment surrounding the system being modeled. This is the approach discussed here. It is a necessary approach when the tests for the model are a complicated function of the environment surrounding the model under test and cannot be derived from the model itself. Such is the case in the signal processing systems that are being modeled in the RASSP program.

One of the goals of the DOD DID for electronic parts is to have the testbench be an immediate interpretation of the specification. Our approach to test bench development is hinged on this idea. Figure 1 below shows the high level test bench generation system that we have implemented. The system specification contains general requirements and specific requirements which together specify the system being modeled. General requirements specify the class of system being modeled. These requirements are the input to a high level design tool which generates VHDL code for the test bench [4]. Specific requirements select a particular member of the class of systems. These specific requirements are considered to be primary. They are fed into a requirements capture tool, such as RDD-100 [5], which uses a math model to derive secondary requirements. Both primary and secondary specific requirements become values for test bench generics.
Figure 1 also shows that general and specific requirements can also be used to generate data files that represent sensor outputs. The general requirements are processed by the Environmental Data Generator to produce the sensor files. These files are read by the testbench during execution using file I/O. The Environmental Data Generator uses tools pertinent to the environment being modeled, e.g. MATLAB [6] and SPW [7] can build certain environmental models from DSP primitives and generate file data from the environmental model, while radar cross section software such as xpitch [8] can generate files of radar return data.

- Nominal range to the center of the swath;
- Pulse repetition frequency of the transmitted signal;
- Bandwidth of the transmitted signal;
- Pulsewidth of the transmitted signal;
- Speed of the aircraft;
- Sampling frequency for the resampling process.

Based on the primary requirements, we can derive secondary requirements, such as the parameters of the signal which are used to do de-ramping, and the sampling frequency.

- Pulse width of the signal used to do de-ramping
  Pulse width of transmitted signal
  + (swath width / speed of light)
  = 30 μs + (375m / 3*10^8(m/s)) = 32.5 μs.

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

- Sampling frequency
  Sampling frequency > 2 * Carrier Frequency
  = 70GHz > 2*33.56GHz.

In this paper we apply this methodology to Infrared Search and Track and Synthetic Aperture Radar applications and show how the specifications for a system can drive the test bench development.

3: Requirements Capture

3.1: Requirements Interface for SAR

The primary requirements of the SAR sensor model are [9]:

- Squint angle (i.e. the radar is pointed at angle γ relative to the velocity vector);
- Carrier frequency;
- Swath width;

Figure 2 illustrates the requirements interface for SAR. This interface can be
implemented using a requirements capture tool such as RDD-100.

3.2: Requirements Interface for IRST

The requirements interface for IRST was developed using RDD-100 [2], which acts as the requirements capture tool that is used to derive the secondary requirements from the primary requirements.

3.2.1: Primary and Secondary Requirements for IRST

Figure 3 depicts the primary requirements fed into the RDD-100 interface, and the secondary requirements derived from the primary requirements based on a math model realized in RDD-100. This information has been provided by the Research Triangle Institute (RTI). Figure 3 shows that the primary requirements for IRST application are:

- **Target Speed** - Speed of the target specified in Machs
- **Platform Type** - The three different types of platforms possible are VF-X, VF, VP
- **Sensor Resolution** - The resolution of the sensor can be either High or Low
- **Revisit Period** - This is the frame update rate, also fed in as a generic to the test bench
- **Range** - This indicates the range of the clutter and target from the platform

The derived secondary requirements are:

- **Target Motion** - This gives the movement of the target in pixels per frame.
- **Clutter Motion** - This gives the movement of the clutter in pixels per frame.

3.2.2: Math Model for the Requirements Interface

The equations used to derive the secondary requirements from the primary requirements are:

- \( \text{Sensor Res Factor} = \frac{100}{1000000} \)
  - if Sensor Res = High
  - if Sensor Res = Low

- \( \text{Platform Velocity} = 448 \text{ m/s} \)
  - if Platform Type = VF-X
  - if Platform Type = VF
  - if Platform Type = VP

- \( \text{Clutter Range} = 1609.344 \times \text{Range} \)

- \( \text{Platform Disp} = \text{Platform Velocity} \times \text{Revisit Period} \)

- \( \text{Clutter Motion} = \frac{2 \times \sin\left(\frac{\text{Platform Disp}}{2 \times \text{Clutter Range}}\right)}{\text{Sensor Res Factor}} \)

- \( \text{Target Range} = 1609.344 \times \text{Range} \)

- \( \text{Target Velocity} = \text{Mach} \times \text{m/s} \times \text{Target Speed} \)

- \( \text{Target Disp} = \text{Target Velocity} \times \text{Revisit Period} \)

- \( \text{Target Motion} = \frac{2 \times \sin\left(\frac{\text{Target Disp}}{2 \times \text{Target Range}}\right)}{\text{Sensor Res Factor}} \)

Figure 4 shows the derivation of the above formulae used for calculating clutter and target motion.

Typical values used for the primary requirements are Target Speed - 1.5 mach, Platform type VF-X, Sensor resolution - Low, Range - 200 miles, Revisit Period - 1 sec.
Total Angular Disp = a1 + a2
\[ \sin a1 = \left( \frac{1/2 \times PD}{\text{Range 1}} \right) \]
\[ a1 + a2 = 2 \times \sin \left( \frac{PD}{2 \times \text{Range 1}} \right) \]
\[ \text{Pixel Disp} = \frac{a1 + a2}{\text{PAFOV}} \]

Figure 4. Derivation of Formulae for Clutter and Target Motion

3.2.3: Realization of the Math Model Using RDD-100

Definitions of terms related to RDD-100 [2]:

- **Behavior diagram** is a graphical representation of the behavior exhibited by a function in terms of the time flow sequence, inputs and outputs.
- **Function** is a part of the system that carries out an action, usually converting an input to an output. For example, in a peach canning system there is a function called Slice Peach, which accepts an Inspected Peach and produces Peach Slices.
- **Item** is something that a function accepts or produces, in other words, an input or output. For example, a peach canning system has an input item called Inspected Peach and an output item called Peach Slices.
- **Discrete Function** represents a function at the lowest level of detail of interest which is not broken down into sub-functions. These accept and produce only Discrete Items.

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

The highest level time function developed was Requirements. The time function Requirements was decomposed into discrete functions - Initialize, Platform Data Init, Platform Data Proc, Clutter Data Proc, Target Data Proc 1, Target Data Proc 2, Target Data Proc 3. The inputs, outputs and the intermediate variables were stored as discrete items, and were declared to be global so that they are available to all the discrete functions. Figure 5 depicts the decomposition of the Requirements time function.

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

- **Initialize** was used to assign values to the primary inputs Target Speed, Platform Type, Sensor Resolution, Range and Revisit Period, and based on the Sensor Resolution the Sensor Res Factor was found.
- **Platform Data Init** was used to initialize the conversion factor mach to m/s. Based on the Platform Type, the Platform Velocity was also initialized.
- **Platform Data Proc** was used to calculate the platform displacement (Platform Disp) based on Platform Velocity and Revisit Period.

Figure 5. Time Function Requirements
• Clutter Data Proc was used to derive Clutter Range using Range, and based on the calculated Clutter Range the Clutter Motion was derived. This discrete function is executed concurrently with Target Data Proc 1.
• Target Data Proc 1 was used to calculate Target Range using Range, and Target Velocity based on the mach to m/s conversion factor and the Target Speed in machs.
• Target Data Proc 2 was used to calculate target displacement (Target Disp) based on Target Velocity and Revisit Period.
• Target Data Proc 3 was used to calculate Target Motion based on the calculated Target Disp, Target Range and Sensor Res factor.

The time function Requirements was verified and simulated using the Dynamic Verification Facility provided in RDD-100. The primary requirements were assigned to the respective discrete items as a part of the Initialize discrete function, and the secondary requirements Target Motion and Clutter Motion were observed.

The event transcript file generated by RDD 100 during dynamic verification contains the derived values of the secondary requirements obtained using the math model. This event transcript file can be parsed using a C program, and the values of the secondary requirements can be extracted and written to a file. As a part of the requirements interface, this file produced by the parser program containing the values of the secondary requirements can be used as input to the Intelligent User Interface (IUI) described in section 6 below.

4: Code Generation

Two approaches are followed to generate testbench code. The first approach called behavioral testbench development uses CASE tools to develop complete high level models of the test bench. The second approach called structural testbench development uses a schematic capture tool to construct the testbench from a library of primitive testbench components. Both the approaches are explained in detail in the following sections.

4.1: Behavioral Approach to Code Generation

A behavioral testbench generation method has several advantages:
• It reduces the test time required for large models.
• The testbenches generated can be used for a class of systems which have the same general functionality.
• It does not require any detailed internal structure information.

A high level approach to testbench development is adopted here. A single entity algorithmic model, specifying the overall functionality of the testbench, is developed for the environment surrounding the model under test (MUT). CASE tools are used to relieve the user of the details of VHDL. I-Logix Express V-HDL [10] is the CASE tool used in this paper to specify the control and data flow behavior of the testbench. Express V-HDL uses statecharts and activity charts for describing control and data flow respectively. Express V-HDL dumps behavioral VHDL code which is finally used for the purposes of simulation.

As examples of this approach, testbenches for two applications are described in this section. Both the testbenches are developed using Express V-HDL. The first testbench has been developed for the Infra-Red Search and Track (IRST) algorithm. This testbench is fully implemented using statecharts which implement control flow. The second testbench is for the Synthetic Aperture Radar (SAR) algorithm and this is implemented fully using activity charts describing data flow.

Statecharts are modified versions of state transition diagrams. They are made up of states and movement from one state to another is represented by a transition arrow drawn from the source state to the target
state. A label of the form `event[condition]/action` is placed on each transition arrow which describes the condition under which the transition is made and describes the action to be taken after the transition. All actions are defined by underlying templates provided in Express V-HDL. These templates contain behavioral code describing the function to be performed by each action. Activity charts illustrate a system's capabilities in terms of its functions (activities) and the information available to these activities. They are variations of data-flow diagrams with labeled arrows representing the information flow.

4.1.1: Modeling Control Flow Behavior - IRST Testbench

The IRST testbench is a typical example of a testbench modeled entirely as a state machine. The whole testbench is implemented in a single overall statechart which describes the entire functionality. The IRST testbench is responsible for providing the IRST algorithm with a sequence of image frames which contain both target and background (clutter) information merged together. In each frame the position of the target is updated, based upon the input parameters like velocity, angle of movement etc.

Figure 6 shows the overall statechart of the IRST testbench. Several of the states in this top level statechart have been decomposed into lower-level statecharts for ease of description and readability. The statechart has two concurrent states, FUNC and CLOCK. The state CLOCK is responsible for generating the clock for the output frame generation. The period of clock that is generated is an input parameter to the testbench. The state FUNC describes the control flow behavior associated with the testbench. The entire operation of the testbench is controlled by the two signals INIT and TRIG. INIT is the initialization signal and default transitions are made into both the concurrent states when it goes high. Clock generation starts when the INIT signal is high. The TRIG signal is used to control the starting and stopping of frame generation. When the TRIG signal goes high, output frame generation starts and when it goes low, frame generation stops. Both the concurrent states are decomposed further into several states.

![Figure 6. IRST Testbench - Top-level Statechart](image)

The state FUNC has the following states:

1. INIT: As soon as the testbench is started, a default transition is made into this state, provided the INIT signal is high. In this state, target and clutter descriptions are translated to arrays. Once all the actions in this state are performed, it exits into the IDLE state.

2. IDLE: No action is taken in the IDLE state. The testbench waits for an event on the TRIG signal to make a transition into the next state which is the RUN state. There are two RUN states corresponding to two modes of operation. Thus, as long as the TRIG signal is low, the output image frames contain just the clutter information.

3. RUN_0: The actions performed in this state are specified in an underlying statechart RUN_0 not shown here. This state is entered from the IDLE state when the TRIG signal makes a transition from low to high and the MODE signal is low. This
state corresponds to the user-interaction mode of operation. Here the position of the target is updated from its previous position based on the velocity of the target and the type of motion. Then the target frame (array) is combined with the clutter array to produce the final image frame. This array is written into a file for application to the MUI. The target update is repeated every clock cycle. This state exits into the FINISHED state when there is an event on the TRIG signal and TRIG goes low.

4. RUN_1 : This state is entered from the IDLE state when the TRIG signal makes a transition from low to high and the MODE signal is high. This state corresponds to the file I/O mode of operation. The output image frames are formed in this state. First the target locations file is read to obtain the new X and Y co-ordinates of the target and it is stored in the target array. Then the target array is added to the clutter array to produce the final image array. This array is then written into an output file. This action is performed for every clock cycle (generated by CLK). The testbench exits into the FINISHED state when there is an event on the TRIG signal and TRIG goes low. All the actions specified in the RUN_1 state are referenced from an underlying RUN_1 statechart.

5. FINISHED : Both the RUN_0 and RUN_1 states exit into this state whenever there is an event on TRIG and TRIG goes low. No action is performed in this state. This state represents the end of the operation of the testbench and no more image frames are produced.

The overall statechart thus represents the description of the full testbench and the entire behavior of the testbench is specified by the top-level and corresponding lower-level statecharts and the underlying behavioral code templates.

4.1.2: Modeling Data Flow Behavior - SAR testbench

The RASSP SAR Testbench should have the capability to produce a sequence of digitized sensor data for the VHDL SAR image processor model. The SAR testbench implemented here is an interactive testbench - some of the input parameters are keyedin by the user. The digitized sensor data are formed by several data-flow blocks embedded in the SAR sensor model. Data flow behavior is modeled in Express V-HDL using activity charts. Figure 7 shows the activity chart used to implement the SAR testbench. The chart is made up of several activities, each of which performs a particular function. The output of these activities are fed to other activities as indicated by the arrows.

The operation of the SAR testbench is explained as follows.

First, the transmitted signal is produced by generating a chirp signal and then multiplying it with a complex tone carrier signal. The received signal is now simulated by just delaying the transmitted signal. Once the transmitted and received signals are generated, these are passed on to the respective down-converters. The transmitted and received signal can be down-converted by multiplying these signals with the complex conjugate of the complex tone used. These two signals are fed to the deramp section where correlation is performed between the received and the transmitted signals. The complex conjugate of the down-converted transmitted signal is multiplied with the down-converted received signal. The output of the deramping section is fed to a decimation section where the number of samples are reduced for purposes of analysis. The output of the decimation section, which is of type real, is converted to bit-vector format and assigned to the final output signals. Thus the output of the SAR testbench model is a sequence of samples that may be stored in a file for further analysis and also for forming a sequence of 40-bit wide words which is fed as input to the SAR processor model. Thus by using
activity charts, the SAR testbench is fully implemented as a data flow testbench.

4.1.3: Generation of VHDL

For both the testbenches developed above, VHDL code can be generated for the purposes of simulation. Express V-HDL uses compilation profiles to define the characteristics and style of the dumped VHDL code. VHDL code can be dumped for several simulators like Synopsys, Viewlogic, Cadence etc. Express V-HDL also has the capability of dumping Verilog code. Another aspect that can be specified using the compilation profiles is the style of code. The dumped VHDL code can be either procedure oriented or in-line oriented. Using these facilities, VHDL code for the SYNOPSYS simulator was successfully developed for both the IRST and the SAR testbenches.

4.2: Structural Approach to Code Generation

4.2.1: SPW Schematic Capture Tool

As described above in section 4.1.2, the SAR sensor model is used to process the input signal, including transmitted signal and received signal, obtained from the radar antenna. It is composed of several basic radar signal processing blocks, such as "Down-converter", "Deramp", "A/D converter" etc. Since SPW is very popular in many companies, it is very convenient to construct a SAR structural model in SPW.

Figure 8 shows the SPW frequency analysis of the output of the SAR model. The distance between the radar and the target
can be calculated from the frequency derivation of the peak position in the frequency domain.

![Frequency Analysis of output data](image)

**Figure 8. Frequency Analysis of the Output Data**

VHDL code from the schematics can be generated from SPW, but SPW can not generate real number VHDL models because it only supports generation of VHDL descriptions of fixed-point architectural hardware designs. Thus we developed our own real number model extraction tool that interfaces with SPW.

In SPW, every structural model has a schematic description file called "$\text{netlist}$". This file contains all the information about the parameters of the blocks, and the connecting information between all these blocks. Therefore, this file is the beginning point for capturing the schematics, it can be used to extract all the necessary information to write a VHDL structural model.

Figure 9 shows the methodology of the SPW schematic capture tool, where the Integration Tool is used to integrate all the structural information, input/output information and parameters captured by shell scripts. It also uses a VHDL support file which is a package containing a library of VHDL procedures corresponding to SPW primitives.

![SPW Schematic Capture Tool](image)

**Figure 9. SPW Schematic Capture Tool**

This SPW schematic capture tool provides an efficient way to generate real number VHDL models automatically.

**4.2.2: SGE-based Structural Testbench Development**

Another approach used in structural test bench development is to create schematics interconnecting the appropriate primitives from a library of primitives using a commercial schematic capture tool like the Synopsys Graphical Environment (SGE) [11]. The schematic editor facility is used to interconnect the symbols created in the SGE symbol editor corresponding to the different primitives. The ports for the overall test bench are specified as a part of the schematic. A symbol is created corresponding to the schematic developed for the test bench. The VHDL interface is used to dump the VHDL code from the symbol created for the overall test bench. The VHDL code dumped for the structural test bench consists of component instantiations corresponding to the different primitives used in the schematic with port maps corresponding to the interconnections between the primitives. The resultant VHDL code for the test bench developed is then simulated using the Synopsys VHDL simulator.

Based on the approach explained above structural test benches were developed for the Synthetic Aperture Radar application.

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

Figure 10. SGE Symbol for SAR Testbench

The genchirp primitive is composed of chirp, compftfp and compmul. Figure 10 shows the symbol created in SGE for the SAR test bench and Figure 11 shows the schematic created in SGE for the SAR test bench. The symbol was created from the schematic. The inputs to the SAR test bench are the chirp bandwidth (BW), the sampling frequency (SAMP_FREQ), the pulse width (PW), the pulse repetition frequency (PRF), the number of samples (N) and the carrier frequency (FREQ).

The outputs of the SAR test bench are the real and imaginary parts of the horizontal component in bit vector format (RBH and IBH), real and imaginary parts of the horizontal component as real numbers (RHI and IRI), the real and imaginary parts of the vertical component in bit vector format (RDV and IDV) and the real and imaginary parts of the vertical component as real numbers (RRV and IRV).

The operation of the model in this schematic is similar to that of the activity chart model in Figure 7.

5. Environmental Data Generator

Comdisco/Cadence SPW, xpatch, or Matlab can be used to generate the data files for the

Figure 11. SGE Schematic for SAR Testbench
model under test. SPW has a primitive called "sink" that can save the data generated in the simulation to a file specified by the user. This data file can be saved as ASCII format so that other testbenches can use it. Also, radar signature prediction software, such as xpatch [8], can be used to generate data files of a radar sensor. It gives a time-domain radar return due to the incidence of a transmitted pulse. MATLAB [6] can be used to generate clutter files for the IRST testbench. A 32x32 two-dimensional array of pixels is produced with the intensity of the pixels ranging from 0 to 255, based on clutter region equations.

6: Testbench Simulation and Control

The test bench model has to be simulated to provide the required stimuli for the model under test. Based on the input information obtained by an Intelligent User Interface (IUI), a simulation control file is created, that is used to simulate the test bench model (either behavioral or structural). Inputs to the test bench can be either user-fed or obtained from a data file. Figure 12 shows the Testbench Simulation and Control System.

![Figure 12. Testbench Simulation and Control System](image)

6.1: Intelligent User Interface

The main purpose of the Intelligent User Interface (IUI) in the above scenario is to configure a testbench that is appropriate to a test or a set of tests, given that all the VHDL components needed for the testbench are available. The system requirements from the requirements capture tool and from other sources can be applied to the testbench model through the IUI. It is also used to generate the necessary input data values, (e.g., target initial position, velocity, angle, mode of motion in the case of IRST application), for the testbenches by either user-interaction or through file I/O.

Besides configuring the testbench with all the input parameters, the IUI also forms the simulation control file for the simulator. This simulation control file is used to create displays during or after simulation of the testbench and controls the overall operation of the simulator. The IUI also serves to forward the testbench outputs to the MUT for the purposes of testing the MUT. The IUI was fully developed in C and runs on a Sun-Os platform.

The IUI either prompts the user with a list of possible selections from which the user has to choose from, or asks the user to key in input values with in a certain range. Figure 13 shows the overall menu structure offered by the IUI. It provides the user with a choice of categories in this menu structure, the options available as a part of the user-driven menu are : application domain - IRST or SAR testbench, type of testbench - behavioral or structural and mode of processing - on-line or post processing.

Based on the input values gathered by the IUI, a simulation control file is formed to control the simulation of the test bench model using Synopsys simulator. The IUI forms the first few lines of the simulation control file consisting of assign statements to assign values keyed in by the user to the input ports of the test bench. The IUI forms the final simulation control file by catenating the above lines with the templates of the simulation control files chosen based on the type of test bench and the mode of processing selected by the user. The templates of the simulation control files are precreated and they control the sequencing of the simulations using the
Synopsys simulator. In the case of on-line processing the simulation control file template used in the simulation control file is used to display results on a Synopsys pad window. In the case of Post processing, the sequence of image frames created by the test bench is fed by the IUI to a MATLAB program that is used to display the image frames in pixel bit map format.

![Figure 13. IUI - Menu Structure](image)

7: Areas for Further Research

Environment-based test benches are most valuable when they act as an executable repository for system requirements. This requires a strong link between the system requirements documents and the test bench components and parameters. We currently have software for finding text patterns in documents. One area of further research is defining search patterns for requirements documents that can find test bench parameter values in text documents. These search patterns would be tailored to specific classes of sensors, and would be associated with the sensor models such as the ones described here. During test bench generation, if a user selects a particular sensor model, he or she could invoke the search pattern on the requirements document to find data values stated there. The use of these extraction tools would assist the user in maintaining consistency between versions of the requirements document and the test benches.

A test bench generator is most valuable when it can interact with a test plan. An area of further research is developing a good framework for system test plans, and then integrating this framework into the test bench generation system. Our goal is a test specification language that could describe succinctly a series of tests. Execution of this specification would cause a particular sequence of tests to be generated and hence a sequence of test benches to be configured and the results combined. This specification language should support black box testing concepts like equivalence class test generation or boundary value test generation, and would derive the appropriate tests from the VHDL data type specifications for external signals.

References: