Chapter 1

Introduction

1.1 Motivation

VHDL (VHSIC Hardware Description Language) is becoming increasingly popular as a way to capture complex digital electronic circuits for both simulation and synthesis. Digital circuits captured using VHDL can be easily simulated and likely to be synthesizable into multiple target technologies, and can be archived for later modification and reuse .

Higher level languages such as C/C++ lack the necessary constructs to model a concurrent hardware system and hence are unsuitable to describe the complex electronic circuits. VHDL allows the behavior of such complex electronic circuits to be captured into a design system for automatic simulation and synthesis.

The astonishing increase in the complexity of integrated circuits and the associated technological revolution bought about the use of CAD tools as a means of reducing the time to market without sacrificing the efficiency and performance. This motivated the research at Virginia Tech to develop an interactive graphics system called '**Modeler's Assistant**' which would help the modeler to develop models with relative ease and enforce a structured approach to modeling. One of the most important aspects of VHDL is its ability to capture the performance specification for a circuit, in the form of a test bench. Test Benches are VHDL descriptions of circuit stimulus and corresponding expected outputs for the given stimulus, that

verify the behavior of a circuit over time. Test benches should be an integral part of any VHDL project, and should be created in parallel with other descriptions of the circuit. These test benches can be applied to a simulator and the same test bench can be used for post-synthesis version of the design.

Once entered into a computer-based design system and a testbench is created, the next objective is to simulate the operation of the model to find out if the description will meet the functional and timing requirements developed during the specification process. The next step in this process is synthesis performed using the design compiler ('dc_shell') provided by Synopsys. The final step in the process of development is checking the compliance between the synthesized and behavioral model to study the timing issues involved.

1.2 Overview

The Modeler's Assistant

One of the more challenging uses of VHDL is creating behavioral models. A critical step would be the selection of the best top-level code partitioning strategy to design a good model. Making a bad decision here can make it difficult or even impossible to succinctly describe the model's functionality - especially the way the parts of the model interact. The Modeler's Assistant relieves a lot of designer's burden in modeling his/her system. This tool accelerates the design of the model by reducing the effort that goes into model by providing a medley of textual/graphical design capture techniques. In this approach, a 'graph' is used to represent the VHDL process graph. The 'nodes' of the graph represent set of processes in the model and 'arcs' denote the signal passage between the process nodes. This graph is constructed by interacting with the designer who then feeds the required input to the tool generated prompts. . The graph forms the basis for delay assignment and test generation. The

Process Model Graph (PMG) shown in figure 1.1 clearly illustrates the structure of a behavioral model.

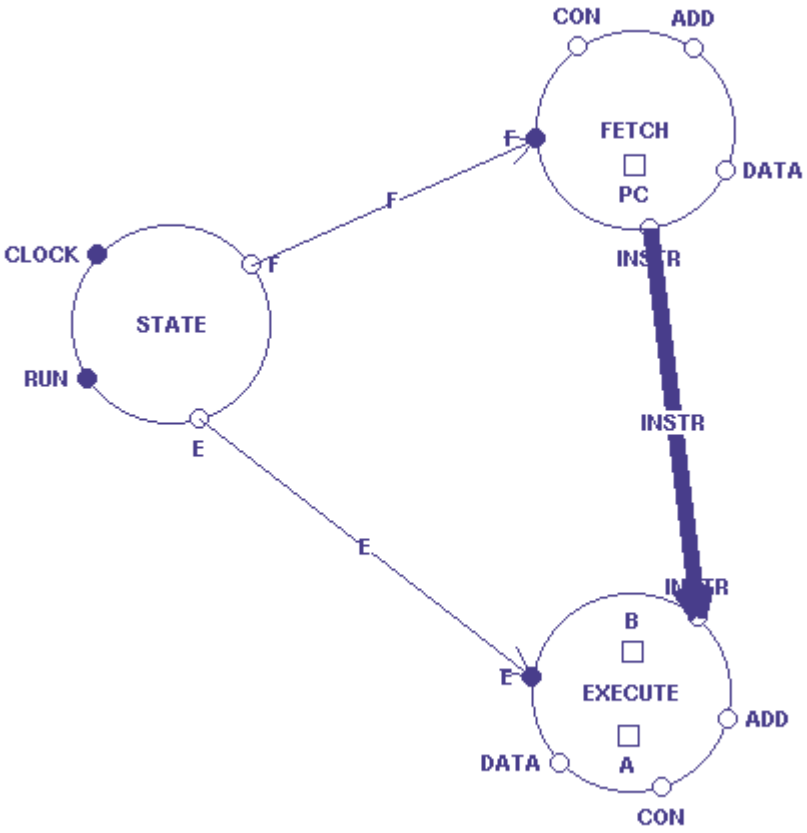

Figure 1.1 A PMG

The necessary input to construct the process model graph (PMG) is entered graphically and textually by responding to the various prompts given by the tool. The tool uses this as the basis to generate the VHDL model and allows the user to graphically put together process modules from a library and it aids in generating a syntactically correct description. After the model is completely generated, the user can generate a structural model of the design.

Advantages associated with Modeler's assistant are

- The model is represented as collection of processes which would then constitute the logical partition of the design under development.
- This type of representation presents the designer a higher level view which hides details.
- This representation aids in understanding the model and makes the overall functionality apparent to the user.
- The tool generates a complete and syntactically correct model.
- This tool provides 'Reusability' at the process level. Frequently used processes are stored in the library and thus significantly reduce the time to develop behavioral models [10].
- Finally, it provides a VHDL learning mechanism.

1.3 Basic Approach to modeling :

"A graphical representation of a behavioral model in which the "parts" of the behavioral model have a clearly defined relationship with one another" is the basis on which this tool is built. This type of representation would aid the user to develop his models accurately and rapidly. The graphically put together processes form the process model graph (PMG). The PMG database is built interactively by selecting a menu item from the menu system. The process database is built by the user by providing information like functionality of the process 'textually'. The tool accesses the PMG database and the process database to construct the VHDL model.

Figure 1.1 shows thus formed process model graph (PMG) which we refer to as ‘Unit’. In this figure, the bigger circle represents the ‘process’, the smaller circle on the circumference of the circle represents ‘ports’[1] used, a rectangle represents a ‘variable’ in the process, a filled rectangle represents a ‘constant’. Variables and Constants can be placed anywhere inside the bigger circle or can be on the boundary of the circle. The port which is ‘sensitized’ is represented by a filled circle. Figure1.2 shows the process model graph as collection of processes. Scalar signals are represented by a line whereas a vector signals are represented by a broad line.

A set of primitives are made available which then aid the designer in the rapid modeling. The user first starts by selecting the “Create” menu item in the Main menu. Then w constructs a ‘process’ by interacting with the tool (described in detail in Chapter 5). The ‘process’ is saved and analyzed by selecting the ‘Analyze’ button in the ‘Process’ menu. The user goes back to main menu and creates a unit by selecting the ‘Unit’ button under ‘Create Menu’. The user adds the necessary ‘processes’ and ‘signals’ between the processes. Once this is done, the user ‘dumps’ the model which would then generate the VHDL. The user can choose ‘Show VHDL’ to view the code. The user chooses ‘Analyze’ in the Unit menu and analyzes the model. Once the model is analyzed, the user can go in for the test bench creation. The user needs to choose ‘Edit Process’ to edit the necessary if any errors are reported. The number of errors would be very few in number when compared to manually generated model.

---

[1]The term ‘ports’ refers to the process port and not as defined in VHDL.

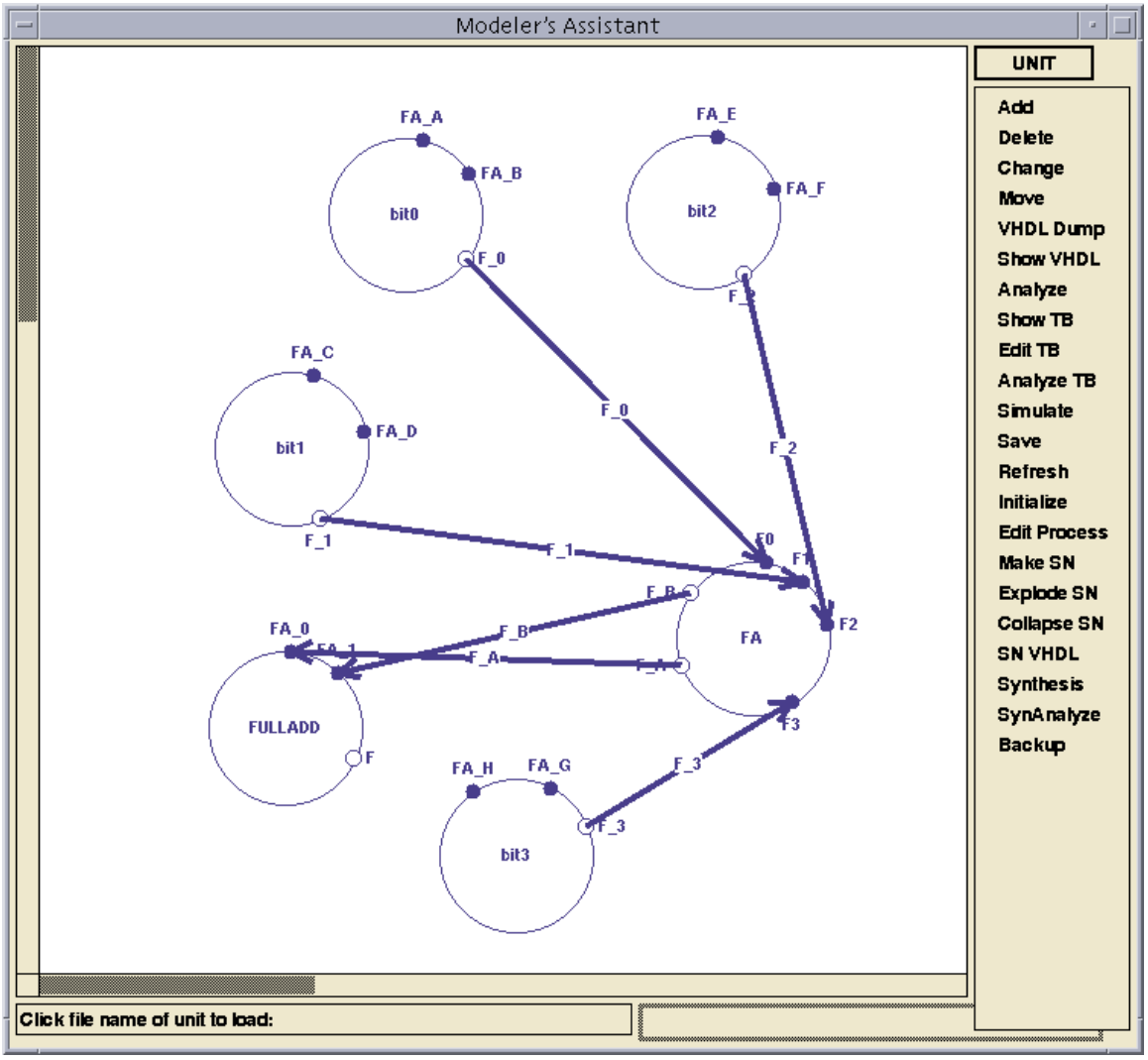

Figure 1.2 The PMG with vector signals.

The user can use the option of creating a ‘Supernode’ if the number of process are high in number. The supernode would represent all the processes by one supernode. In this tool, a supernode is represented by a circle filled in with slant lines. The internal signals are not shown. ’Collaspe SN’ collects all the process to make a supernode. ‘Explode SN’ explodes the supernode into processes again. This would represent a complex model in a simple way thus avoiding a cumbersome representation. The user can choose to generate a structural or a behavioral model at ‘supernode’ level. Thus the tool can help user to generate a structural model .

Chapter 2

Previous Limitations and Enhancements

2.1 Previous Work:

A software tool, called Modeler's Assistant was developed at Virginia Tech. Modeler's Assistant, abbreviated as 'MODAS' accepts 'process model graph' (PMG) as input and generates the VHDL code. The tool was developed in the Unix programming environment.

There were several limitations in the version 4 and bugs in the software which constrained the effective usage of this tool. The bugs are corrected in current version, that is, Version 5 of Modeler's Assistant.

Some of the limitations were:

1.The tool lacked the very important data type, the STD_LOGIC that is now the de facto standard in the industry now.

2.The tool lacked interfaces to simulation and synthesis, features without which the tool cannot be really useful. Any attempt to model a real system would be incomplete without checking for functional compliance and synthesis.

3.The executable ‘mvhd’ used to create the dummy entity for the process was lost and therefore the ‘process’ couldn't be analyzed.

4. When the ‘unit’ was analyzed, it lacked the ‘spc_elab’ switch which enables one to check whether the written VHDL model is synthesizable, that is, the synthesis compliance check is performed for the VHDL against the Synopsys synthesis coding rules.

5. The test bench for the model was not developed fully and did not have the necessary constructs in it. The test bench generation software had to be rewritten completely in this version.

6. The test bench had to be deleted by choosing ‘TestBench’ under the ‘Delete Menu’. This would amount to the inefficient use of the tool. If the user makes mistakes while manually entering the test vectors, the whole test bench had to be regenerated. Now the user can choose to edit the text manually or can regenerate the test bench again.

7. The test bench didn't include any generic mapping statements or configuration statements which would aid in reusing the same test bench for the testing the synthesized file obtained after the synthesis step.

8.If the analyzed file results in errors then the ‘Modeler’s Assistant’ doesn’t show a file with line numbers which would match the line numbers in error messages. So if the source file consists of hundreds of lines, the user needs to manually count lines to know where the error occurred. This disadvantage has been taken care off in the version 5 of ‘Modeler’s Assistant’.

9.The package declarations were made only after the complete model was constructed and not at the “process” level where the processes are created. This would cause inconvenience in using certain type declarations at the process level. This feature has now been added.

10. Some of the menus didn’t allow the user to backup to the previous menu. Now the user can exit any of the menu by selecting the ‘Backup’.

2.2 Current Version of the Modeler’s Assistant:

The version 5 of Modeler’s Assistant enhances already existing features, removes bugs associated with the tool and adds new features.

The environment of the Modeler’s Assistant provides us with

1. Behavioral Modeling
2. Automatic test bench generation providing a range of test benches for the user to choose.
3. Interface to synthesis tools
4. Validation of the behavioral model alone and validation of the behavioral model versus the synthesis model.

The figure 2.1 shows the block diagram of Modeler’s Assistant. The architecture shows that they are two main databases called the PMG database and process database. The process model graph data base stores information regarding the geometry of the process model graph, that is, the coordinates of constructs like process, variable, signal etc. It also stores information regarding the type of the ports, variables, whether the particular port is in the sensitivity list etc. The user can edit this database by using a PMG editor. The user is free to change the geometry, element names, their types etc. The functionality of the process which is entered textually is stored in the process data base. The information in this database can be manipulated by using process functionality editor, which in this case are the text editor and the text parser.

The next main component is the primitive generator which creates the process primitives and stores them in a library.

The VHDL generator takes information from the PMG data base and the process database to construct a complete VHDL model. This generated model is analyzed by invoking the 'analyzer' tool of Synopsys. The test bench generator accesses the two central databases and generates the test bench. The test bench is also analyzed by invoking the analyzer from Modeler's Assistant and piping back the results onto the display window.

The 'Simulator' which in this case is the 'Synopsys graphical debugger' is invoked to simulate the model. The model is simulated to check for its compliance with the desired behavior. The next step would be to synthesize the model to map it to a desired library. Modeler's Assistant spawns off a child process using the UNIX system call fork(). In the child process, the dc_shell is invoked on an X- terminal. Once the synthesis is done, the synthesized VHDL model is analyzed and simulated. Validation is now performed to check the compliance of the behavioral and synthesized model. This can be achieved by choosing the 'Validation' type of test bench. The model is then simulated and the results of simulation would help the user understand the timing issues involved.

A structural model of the design can be obtained by making use of 'supernodes'. A 'supernode' is defined to be collection of 'processes'. VHDL generator generates this model by accessing information from the two databases.

The corrected Limitations in this version of Modeler's Assistant:

- The user can use STD_LOGIC data type. That would remove the disadvantage associated with the data types provided to the user. The tool provides the user with a menu driven format for selecting the packages. User can choose any of the IEEE packages like IEEE.std_logic_arith, IEEE.std_logic_textio etc. The user can also add any of user defined packages or select the two packages provided by this tool called the VHDLCAD and USER_TYPES. More of this is discussed in the Chapter 3.

- The VHDL code outputted by ‘Modeler’s Assistant’ can be analyzed, with or without, synthesis switch by selecting the 'Analyze' from the menu. For synthesis purposes, an additional switch is added to the ‘Analyze’ option which would further check whether the model is synthesizable.
- The major changes are those relating to test bench generation. Now the user can choose from a range of test bench options to suit the requirements. This menu driven system provides the user from a simple test bench (called Shell in the menu) to a test bench which would check for compliance between the behavioral and synthesized models. There is a test bench for testing combinational circuits where one can generate all possible combinations of a select few inputs to test the model. If the circuit is a clocked one, the user can choose the ‘CG +OSC ‘ type of test bench, wherein the ‘oscillator’ would generate the clock of desired frequency and ‘combinational generator’ would generate all possible combinations of select few inputs. Models can be driven from file I/O, and therefore we have provided ‘Modeler’s Assistant’ with this capability too. The test vectors for the test bench can be automatically generated by language independent timing diagrams drawn by the user. This uses the Synapticad software.

Simulation and Synthesis features are added to complete the development process. The code can then be simulated to verify the compliance of the design behavior with the desired functional behavior. This can be achieved by selecting the 'Simulate' menu. Then the model can be synthesized using the design compiler provided by Synopsys by selecting the 'Synthesis' menu item. . The user needs to write a script file which is used as an input to the synthesizing tool. ‘Synopsys’ tools like the graphics debugger and dc_shell are used to perform simulation and synthesis. This is discussed in detail in later sections.

2.3 Implementation Details:

The current version of the Modelers' Assistant has been developed on Sun Solaris workstation.

The majority of the code was done on the Sun Sparc worksation. The first version which was developed on a PC running on Linux operating system[2]. The Sun source browser[3], Sun Dbx tool were mainly used to search through the code and for debugging purposes. These tools proved very useful in the development in writing the software for the tool and to fix the bugs of the old code. The current version has approximately 20,000 lines of source code. Therefore without these tools it would have been a laborious task.

Modeler's Assistant uses Athena libraries, X and Xt libraries and the C libraries. The Athena widget library is extensively used to setup the required windows and menu systems. The X and Xt libraries were used to provide the look and feel of the graphical interfaces. The X window systems[7] is a hardware independent system developed at MIT and Digital Equipment Corporation. The X takes 'mouse' as the input to trigger events. The purpose of Xt(X toolkit) is to provide an object oriented layer that supports user-interface abstraction called widget[7]which are mostly predefined. The widget acts independent of the application and can be used several times. Widgets provide a simplified approach to create menu systems and graphics. [7] discusses these ideas in detail.

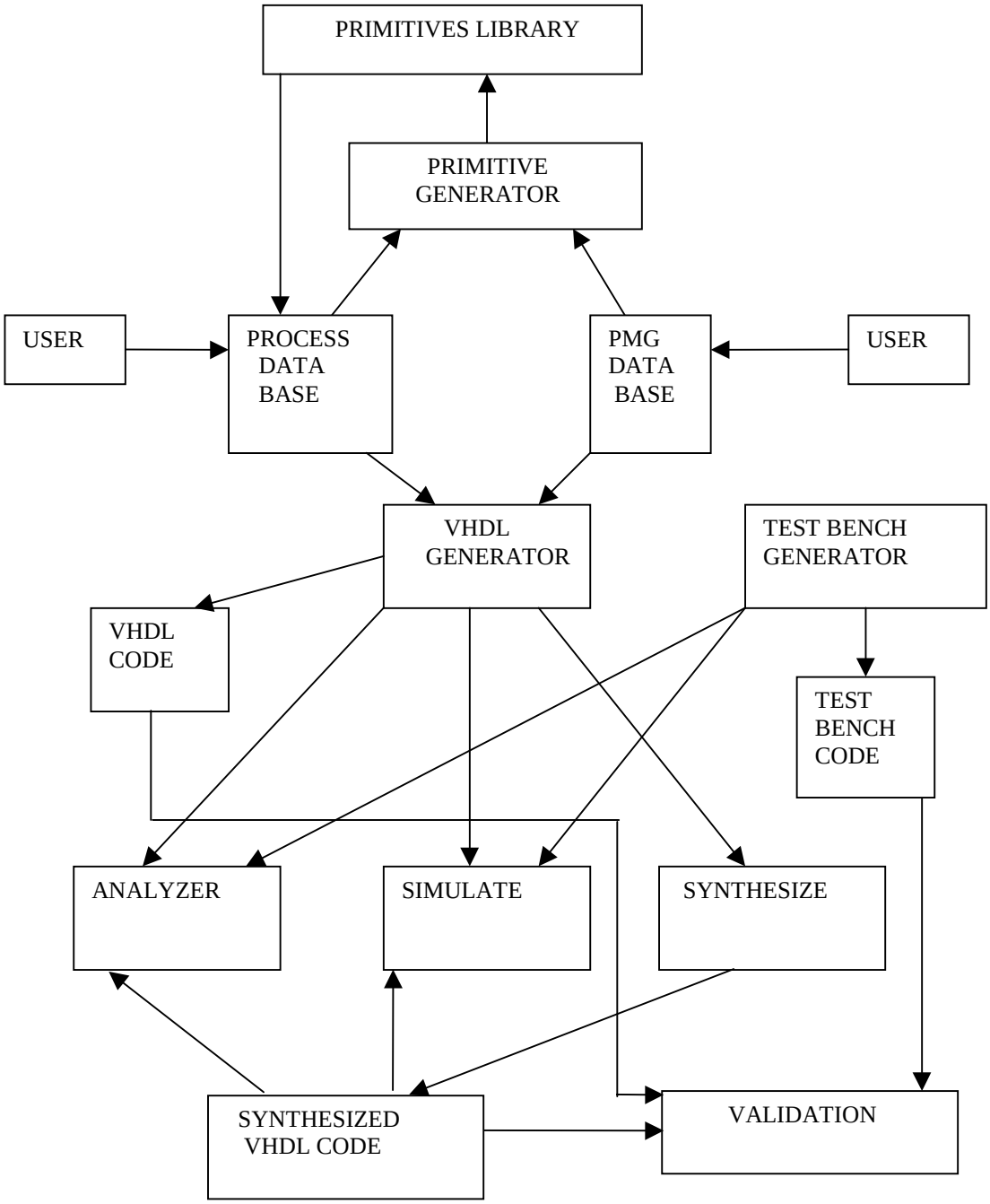

Figure 2.1 Block Diagram of Modeler's Assistant

Chapter 3:
Packages

3.1 Packages Overview

VHDL is a complex language to model hardware and has many data types which would specify what values the object may have and the type of operations that could be performed. Integer, Bit and their vector partners, Real, STD_LOGIC and their vector partners etc to name a few.

In modeling a system, often we use same type declarations and it would be tedious to repeat the declarations each time they are used. In VHDL we have “package” where we could declare these and then declare this package in the VHDL source code. The declarations are made visible by referring to the package name. The package STANDARD contains definitions of BIT, INTEGER, TIME etc. STD_LOGIC data type which is widely used in the industry is defined in the IEEE packages. Various conversion routines and operations are defined in these packages.

The packages to be added when Std_logic type is used, are the standard IEEE packages. These packages are IEEE.std_logic_1164, IEEE.std_logic_arith, IEEE.std_logic_components, IEEE.std_logic_signed, IEEE.std_logic_textio etc. The IEEE.std_logic_1164 is the basic IEEE VHDL package which declares the basic values and functions for the IEEE 9 valued system. Other packages provide overload operators and other useful capabilities.

A “use” clause preceding a unit will make the items declared in the package declaration visible in the unit. The ‘use’ and the ‘.all’ extentions tells the compiler to use all the elements or conversion routine or functions contained in the IEEE.std_logic package. These packages are analyzed so no analysis is required by user. The user can declare these packages above the entity declaration region. The packages are declared as shown in figure 3.2.

To add packages in Modeler’s Assistant, choose the ‘Add’ menu item in the ‘process’ menu. When the user selects ‘Add’, it gives an option of adding a ‘package’. Choose the ‘package’ menu item and this leads the user to package menu wherein the user can click on any of the listed packages. If the user needs to enter a user defined package then the user needs to select ‘UserDef’ and enter the name of the package. Thus the user adds the packages to the PMG database. The tool extracts information from this database to add the packages when constructing the VHDL model.

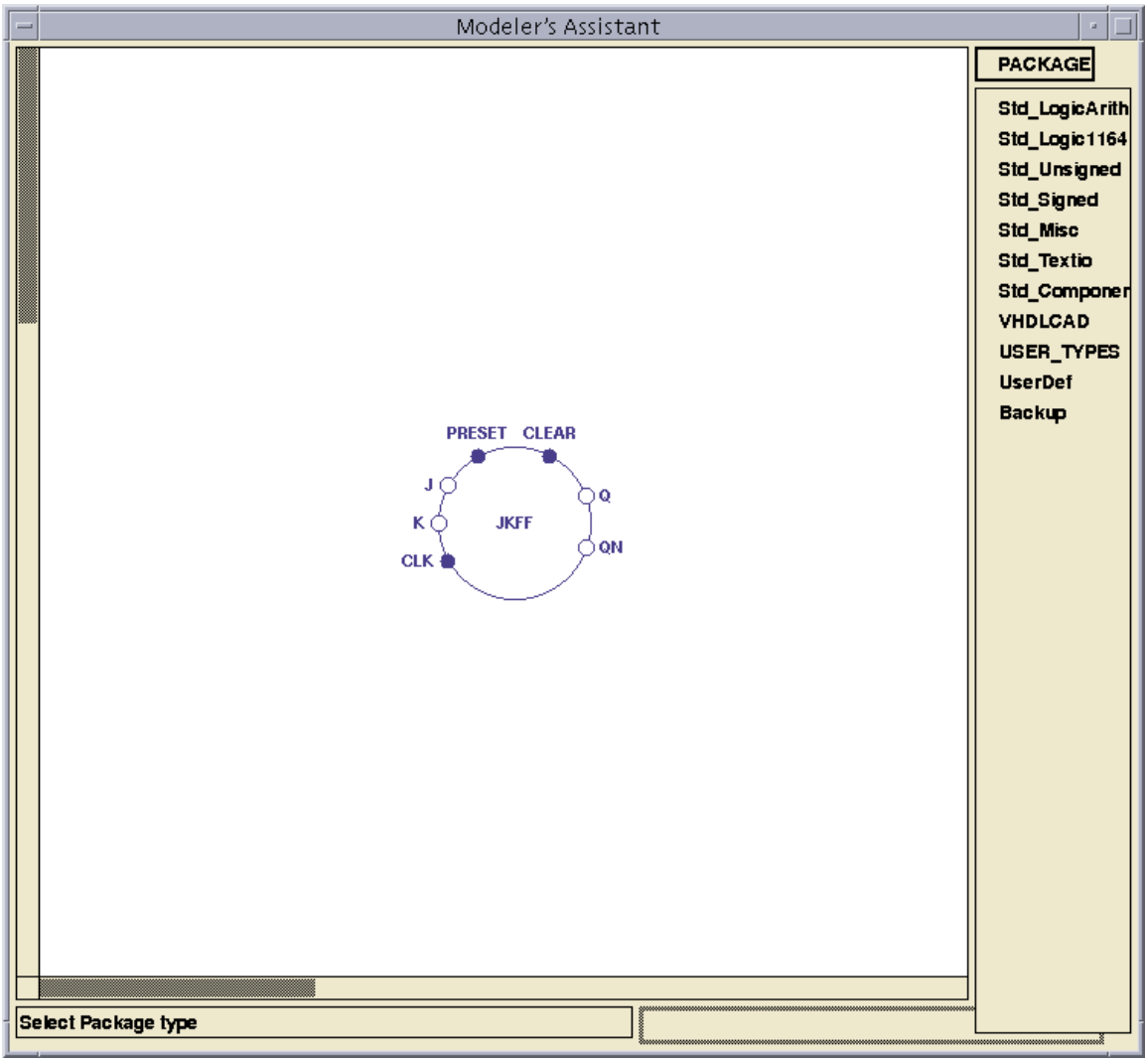

Figure 3.1 The Package Menu in Modeler's Assistant

Definition of a general package[1]

```
package finc is
function INC(X:BIT_VECTOR) return BIT_VECTOR;
function DEC((X:BIT_VECTOR) return BIT_VECTOR;
end finc;

package body finc is
begin
   function INC(X:BIT_VECTOR) return BIT_VECTOR;
    ----
    ----
   end INC;

  function DEC(X:BIT_VECTOR) return BIT_VECTOR;
    ----
    ----
   end DEC;
end finc;
```

Declaration of package in VHDL:

```
use work.finc.all;
use library IEEE;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_1164.all;
entity one is
  port(-----);
end entity;
```

Figure 3.2 Showing the Definition and Declaration of Package.

Chapter 4
TEST BENCH CREATION

Testing is a vital aspect of the model development process in which stimuli are applied to the inputs and the corresponding outputs are checked to see whether they simulate the desired behavior. Such verification is desired in the process of model development before prototypes are built because it would be quite difficult to find bugs in silicon and thus would result in a poor and time consuming process.

In VHDL, a test bench provides the necessary input stimuli to the inputs and aids in the observation of the response to the applied inputs. However, manual test generation is a tedious and time consuming process.

To address these needs, here at Virginia Tech, we have tried to automate the process where the designer is relieved the details of the test bench.

In this version of the Modeler's Assistant, we generate a user defined test bench which contains declarative sections like entity, architecture, component instantiation and signal declarations. The PMG and the process databases have all the information stored are used to make a test bench template. When the appropriate test bench is chosen, the user is prompted for various information. The various types of test benches help the user to provide the input stimuli in different ways. The input can be manually entered or can be generated from a

combinational generator (oscillator, if the input is a clock) or can be read from a file. The tool also provides the capability to draw the waveforms of the inputs instead of textually inputting them to the tool. The same test bench template is used for the post synthesis model too. This test bench is used for the validation purpose, that is, checking the compliance between the behavioral and synthesized model. Any non compliance is reported while in simulation stage. Thus, the user can get an idea about the issues involved with delays between gates when the circuit is at gate level. The types of test benches are discussed in the coming sections.

4.1 Various Test Benches

After the completion of model when the user desires to test his model, then the user needs to select 'Edit TB' menu in the 'Unit Menu' and this would lead to a child menu ,where the user has options to select one type of test benches. A prototype of a simple test bench is shown in figure 4.1.

Shell:

'Shell' is the test bench template in which the entity, architecture, component and configuration declarations are automatically generated by accessing the data from the PMG database. The user needs to enter the test vectors manually and the test bench is complete and syntactically correct. Since the user just needs to enter the test vectors, the time to generate the test bench is drastically reduced. Figure 4.1 shows the 'Shell' test bench as generated by the Modeler's Assistant. The user needs to enter the vectors at the appropriate place.

```
library IEEE;
use IEEE.std_logic_1164.all;
    entity BUFF_REG_TEST_BENCH is
    end  BUFF_REG_TEST_BENCH;
architecture TESTBENCH of  BUFF_REG_TEST_BENCH is
    signal D0: BIT_VECTOR(1 to 8);
```

```
    signal DI: BIT_VECTOR(1 to 8);
    signal STRB: BIT;
    signal NDS2: BIT;
    signal DS1: BIT;
--/**component declaration**/
component BUFF_REG
  port (D0: out BIT_VECTOR(1 to 8);
          DI: in BIT_VECTOR(1 to 8);
         STRB: in BIT;
         NDS2: in BIT; DS1: in BIT);
 end component;
begin
        UUT1:  BUFF_REG
       port map(D0, DI, STRB, NDS2, DS1);
--/**Enter the TEST VECTORS**/
  process
       begin

 wait;
end process;
end TESTBENCH;
configuration CFG_BUFF_REG of BUFF_REG_TEST_BENCH is
   for TESTBENCH
        for UUT1 :BUFF_REG
          use entity work.BUFF_REG(BEHAVIORAL);
       end for;
  end for;
end;
```

Figure 4.1 Showing the simple ‘Shell’ Test Bench:

Combinational Input Generator:

If the user is modeling a combinational type of circuit wherein he needs to test the model extensively for a particular set of inputs then this type of test bench is useful. In this test bench, we have a combinational generator which would generate all possible combinations for the desired inputs. The user is prompted for the names of the input whose varied combinations needs to be applied to the circuit for successful testing. Consider a model under test which is to be tested for all possible combinations of signals, lets say, A , B, C, D. Therefore the user needs to assign the pulse generated outputs(called PG(0) to PG(3)) to one of the four inputs, A , B, C, D. Thus the combinational generator would generate 16 possible combinations. The following figure 4.2 would show a pictorial representation. Thus the user job is reduced since the only test vectors to be entered manually are that of RST and EN.

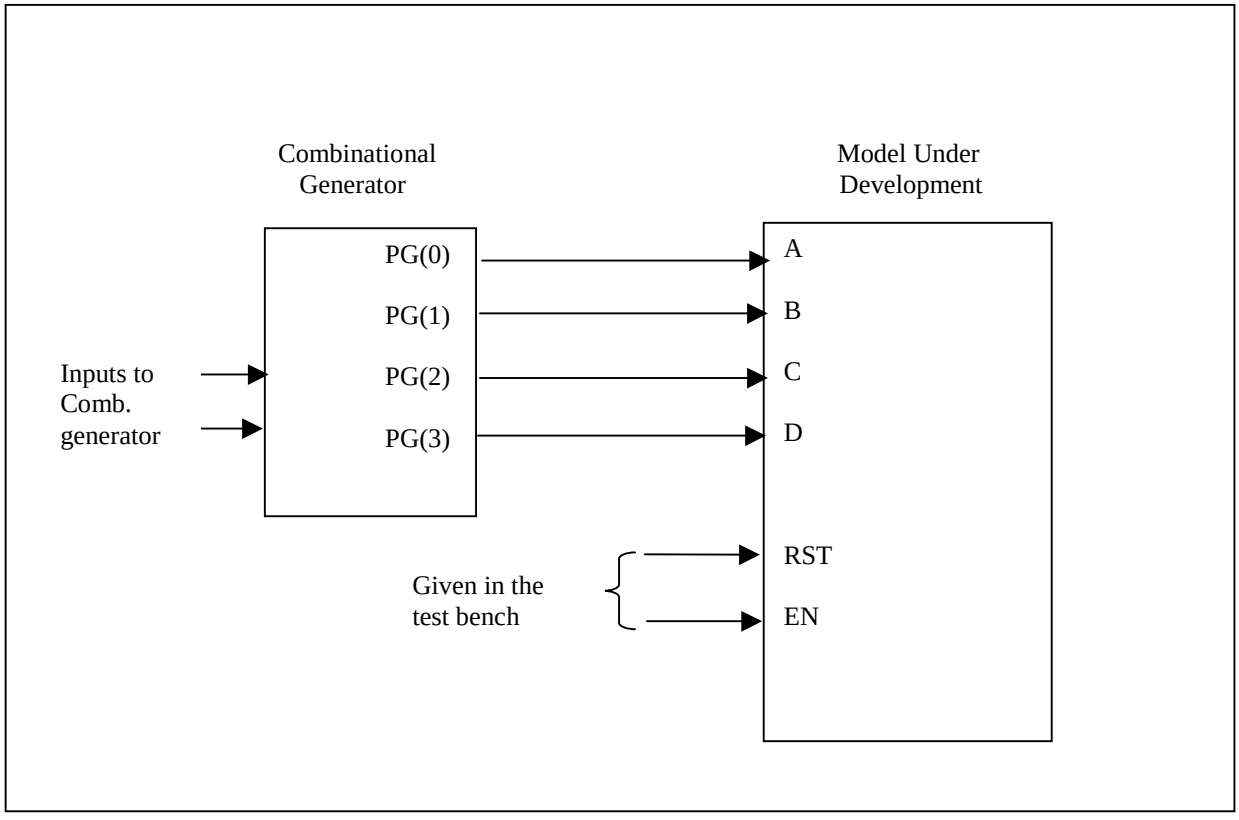

Figure 4.2 Showing the CG Test Bench Block Diagram

CG + OSC:

This test bench is best suited for clocked sequential circuits. If the model under test is a JKFF flip-flop and the user desires to test the model extensively for all possible combinations of J, K, S, R, then this is the test bench type to be selected. The 'OSC' primitive which is declared in the component declaration would generate the clock of desired frequency.

Here the clock model is an oscillator where the clock characteristics are specified in terms of 'HITIME' and 'LOTIME'. The user is prompted for the duty cycle in terms of 'HITIME' and 'LOTIME'.

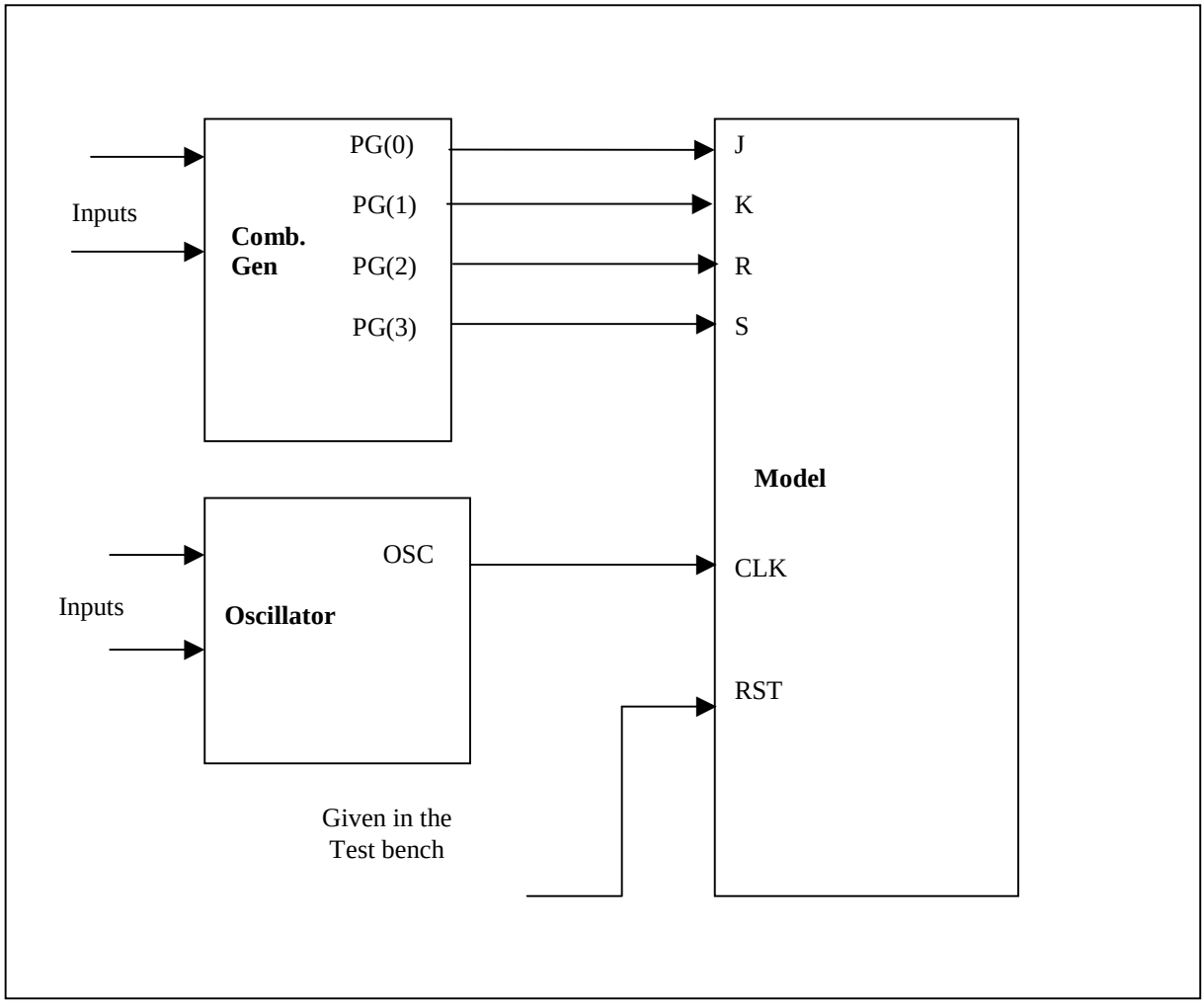

Figure 4.3 Showing the CG+OSC Block Diagram

The input generator would generate the necessary combinations for the inputs.The user has flexibility to map any of the signals to any of the bits of the combinational generator output bits. Thus the user can map J to PG(0) or PG(1) or any other bit to suit convenience. Therefore the only test vector to be entered manually would be 'RST'. Thus a lot of time is saved for the user. The period between the changes from one combination to the other is specified in terms of generic called 'PER'. This generic value should at least be equal to the clock frequency else the model simulation wouldn't be correct.

FILE I/O:

VHDL has this facility wherein the test vectors can be read from a file and written to file.All files are data steams and are sequential in nature [1] and are read during the simulation. The declarations of this text I/O are declared in package STD.TEXTIO. The data in these files is organized in lines and can point to variable number of elements. Refer [1] to study the details. In this tool, the user is prompted for the number of vectors to be read, their type, that is, scalar/vector and if a 'vector', the user is prompted for the size. Then the user is prompted for the signal to which these vectors correspond to. The tool then generates the complete test bench. The prototype is presented in figure 4.4.

```
use work.all;
use std.textio.all;
  entity TEST_BENCH is
  end  TEST_BENCH;

architecture TESTBENCH of  TEST_BENCH is
    <--signal declarations……>

component MODEL
  port ( ……);
end component;
```

```
begin

UUT1:  MODEL
    port map(……);

--/**Enter the TEST VECTORS**/
  process
       begin

wait;

  end process;
--/**FileIO Statements**/
process
      variable VLINE: LINE;
      variable V1:BIT;
      variable V2:BIT;
file INVECT: TEXT is "temp.txt";
  begin

   START_PLAY <= '0', '1' after 3 ns;
   wait on START_PLAY until START_PLAY = '1';
    while not(ENDFILE(INVECT)) loop
    READLINE(INVECT, VLINE);
    READ(VLINE, V1);
    READ(VLINE, V2);
    FooA <= V1;
    FooB <= V2;
    wait for 1 ns;
  end loop;
end process;
end TESTBENCH;

configuration CFG_MODEL of  TEST_BENCH is
   for TESTBENCH
        for UUT1 :BUFF_REG
            use entity work.MODEL(BEHAVIORAL);
      end for;
```

```
  end for;
end;
```

Figure 4.4 Showing the ‘FileIO’ Test Bench.

FileIO + OSC:

This is useful for clocked sequential circuits wherein the clock is generated by the ‘OSC’ primitive and the other test vectors can be read from the file. This facility is now available only for objects having BIT/BITVECTOR data type.

Validation:

There are gate delays in a synthesized model. No such gate delays exist in the behavioral model. Therefore it would be advantageous to study the timing issues when the model is synthesized. The test bench can be used to simulate both the behavioral and synthesized model simultaneously for checking the compliance between the various signals of the same model. The signal in the synthesized model is given an ‘_S’ extension.

The user is prompted to enter ‘y’/’n’ for the signals in the model and signals for which the response is ‘y’ are the signals which are checked. This uses the PMG database for information regarding the entity, architecture, signal and component declarations. The configuration part consists of two configuration declarations, one for the behavioral model and the other for the synthesized model. Figure 4.5 illustrates this test bench type.

```
library IEEE;
use IEEE.std_logic_1164.all;


entity TIMECKT_TB_VAL is
end  TIMECKT_TB_VAL;


architecture TESTBENCH of  TIMECKT_TB_VAL is
    signal C: STD_LOGIC;
    signal RESET: STD_LOGIC;
    signal CLK: STD_LOGIC;
    signal B: STD_LOGIC;
    signal A: STD_LOGIC;
    signal EN: STD_LOGIC;
    signal F,F_S: STD_LOGIC;
    signal CHECK_EN :BIT;
--/**component declaration**/
component TIMECKT
  port (C: in STD_LOGIC;
        RESET: in STD_LOGIC;
        CLK: in STD_LOGIC;
    B: in STD_LOGIC;
    A: in STD_LOGIC;
    EN: in STD_LOGIC;
    F: out STD_LOGIC);
end component;


begin

UUT1:  TIMECKT
     port map(C, RESET, CLK, B, A, EN, F);
UUT2:  TIMECKT
     port map(C, RESET, CLK, B, A, EN, F_S);
```

```
--/**Enter the TEST VECTORS after begin**/

process
    begin
      RESET <= ’1’ , ’0’ after 10 ns;
      CLK  <= ’0’, ’1’ after 20 ns, ’0’ after 25 ns, ’1’ after 30 ns,
              ’0’ after 35 ns, ’1’ after 40 ns, ’0’ after 45 ns,
              ’1’ after 50 ns,’0’ after 55 ns,’1’ after 60 ns;
      EN   <=  ’1’, ’0’ after 70 ns;
      A    <= ’1’,  ’0’ after 25 ns,  ’1’ after 35 ns;
      B     <= ’1’;
      C     <= ’1’,  ’0’ after 35 ns,  ’1’ after 45 ns;
 wait;
  end process;
end TESTBENCH;

CHECK_EN <= ‘1’ after 2 ns;
process
 begin
      wait until CLK = '1';
if (CHECK_EN = '1') then
          assert ( F = F_S )
          report "Validation Failed";
  end if;
end process;

configuration CFG_TIMECKT of TIMECKT_TB_VAL is
   for TESTBENCH
      for UUT1 :TIMECKT
      use entity work.TIMECKT_BEH(BEHAVIORAL);
      end for;

      for UUT2 :TIMECKT
    use entity work.TIMECKT(SYN_BEHAVIORAL);
      end for;
  end for;
end;
```

Figure 4.5 Showing the ‘Validation’ Test Bench

When this model is simulated by using this test bench the simulation shows the results from which we can estimate the amount of delay involved between the behavioral and synthesized model. It would aid the designer to fix his timing parameters accordingly. Thus this type of test bench is very useful for analysis purposes. Whenever the value check fails, an error is reported.

Syncad:

Synapticad, a company based in Blacksburg, VA has developed a tool called 'TestBencher Pro' which is basically a waveform generator from which the corresponding VHDL code is generated. The user draws the waveforms which are language independent. This would provide a clear visual picture of how the waveform for the inputs should look when simulated. The user needn't enter the test vectors manually but would draw them and save the file in the current working directory where 'modas' executable and other related files are present. The user needs to select 'SHOWTB" in the 'Unit Menu' and has to enter the name of the file when prompted and then the Modeler's Assistant generates the test bench file with these vectors in it. Thus the time to generate a test bench would be drastically reduced.

This newly generated test bench is saved with '_tbench.vhd' extension. Thus the user has flexibility to choose between the test bench which has no test vectors added to it and one which consists of the test vectors.

4.2 Analysis

The VHDL code generated by the tool needs to be checked for syntax errors. These analyzer tools check the code for these errors and report them so that the user can generate a complete, syntactically correct code. This is the first step undertaken to check the design. All the code generated by 'Modeler's Assistant' is analyzed using the Synopsys Analyzer.

Analyzing a Process/Unit: The user creates a ‘process’ and the tool uses the information from the PMG data base and generates only the architectural body of the code and doesn’t generate the entity. Therefore a dummy entity is created and then the process is analyzed. An executable ’mvhd’ is used to generate this dummy entity and then the process is analyzed. Any errors reported at this stage need to be corrected.

Once all the processes required for a model are being created, then the user proceeds to create a ‘Unit’ which would then represent be the whole design. The user dumps the unit which would generate the VHDL code. Then the user analyzes the code for any syntax errors. The errors reported would be very few in number because the tool automates the process of generation of VHDL. The user is prompted, if the analyzing should be done with ‘synthesis’ switch on. If the user chooses to say ‘n’ then the code is analyzed with ‘-c’(coverage) switch on. This would generate ‘.o’ extension file which represent the compiled code. This file is generated in addition to the ‘.mra, ’.sim’ extension files. If the user chooses ‘yes’ to the synthesis switch then, an additional switch is added as the analyzing option which would further check whether the model is synthesizable. This switch would invoke the Synthesis Policy Checking(SPC). The analyzer compares the source files to the VHDL subset supported by the VHDL Compiler. Before a file is analyzed, all the files it uses should also be analyzed with the -spc_elab option[4]. This is because the analyzer creates extra design library files when it checks synthesis policy. These files have the .syn suffix. The errors and warnings wouldn’t affect the creation of simulation design library files, or the total error and warning count reported by the analyzer[4]. These are used to inform the user why the model cannot be synthesized. If the VHDL model is written adhering to the synthesis tool , it could be synthesized to obtain a gate level model Thus the VHDL is analyzed with various options.

Once the PMG is successfully analyzed, then the user needs to create the test bench and analyze the test bench. Figure 4.6 shows the analyzing window when the test bench of the PMG is analyzed.

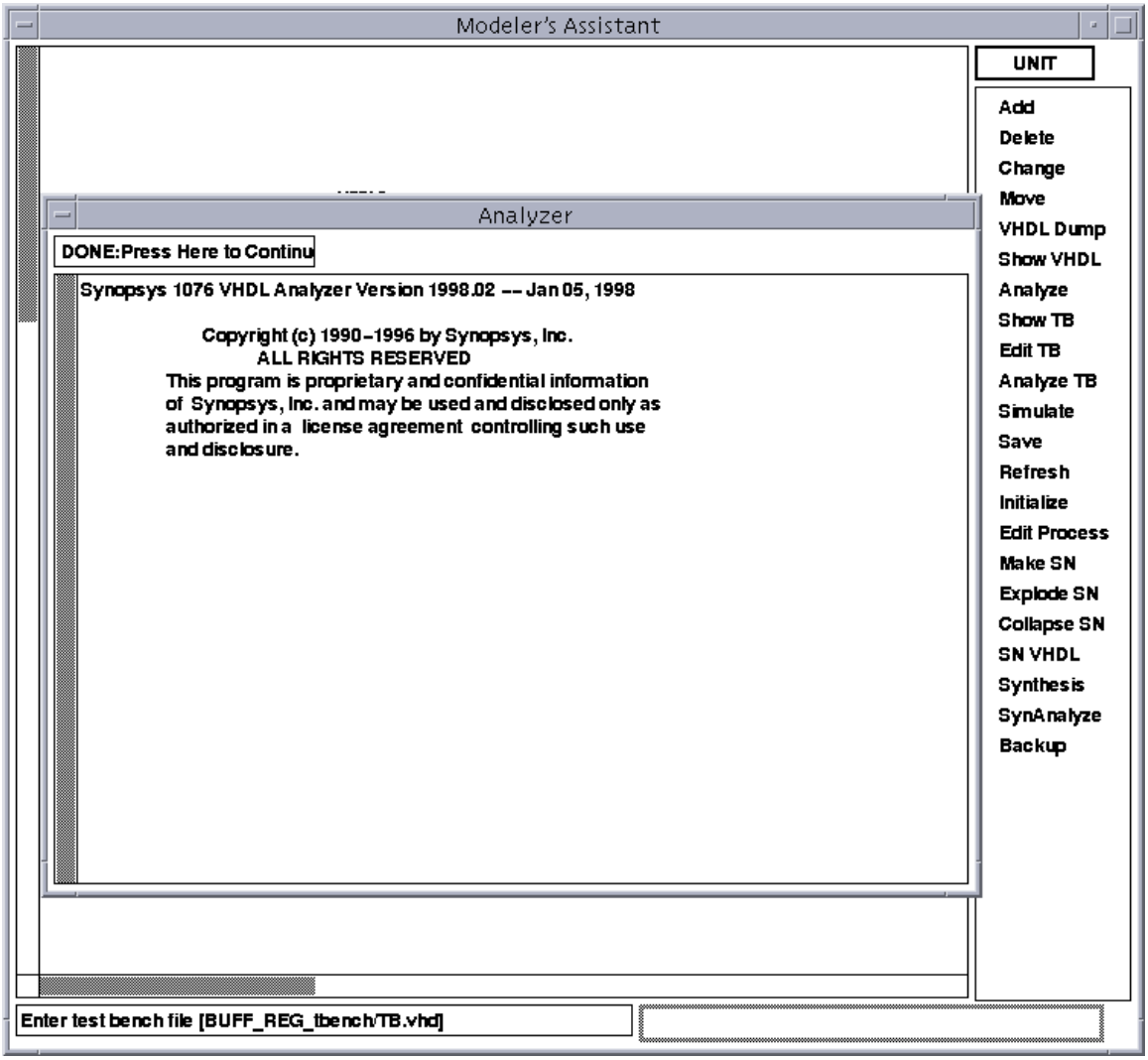

Figure 4.6 The Analyzer Window

4.3 Simulation and Synthesis

To complete the development process, simulation of the design needs to be performed. Therefore once the generated code and test bench code are analyzed with no errors reported, then the user can step ahead to check the functional behavior of the model under development

by choosing ‘Simulate’ button in the ‘Unit’ Menu. This would open the ‘Synopsys Graphics Debugger’ and then the required signals, variables are traced and the simulation is run and then the waveforms are analyzed for the any discrepancies in the behavior. If the design complies with the functional behavior the designer can proceed to synthesis. But if any discrepancy occurs, then the user needs to fix it by quitting from the simulator and going back to Modeler’s Assistant to perform the necessary changes.

As the end of the development cycle, the user can select ‘Synthesis’ to synthesize the model. The tool prompts for name of the script file which is used to synthesize the model. The script file required is as shown below. For more information, refer [4].

```
analyze -format vhdl <design name>.vhd
elaborate <entity name> -architecture BEHAVIORAL
create_clock -period 10 -waveform {0 5} <Clock name>
compile
write -format vhdl -output <new design name>.VHD
write -output <new design name>.db
report_timing > <new design name>.rep
report_area >> <new design name>.rep
```

Figure 4.7 Showing the Design Analyzer Script File

Once the synthesis is finished, then ‘.rep’, ’.vhd’ files are created as a result of the synthesis. The ‘.rep’ file contains the report regarding the timing and area parameters of the model. The ‘vhd’ file which is the synthesized VHDL is analyzed and the same test bench is used for the post synthesis simulation.

Chapter 5
Programming Manual

This chapter mainly concerns with how to model the design using the Modeler's Assistant. It gives a detailed procedure and would aid the user to understand the tool.

5.1 Development Process:

Step 1: MAIN-CREATE/ PROCESS

The user creates a 'process' and therefore starts to build the PMG. The first information to be given is the 'name' of the process. The user needs to give this information when he/she selects 'Process' from the Create Menu.

- *Adding Ports/variables/Constants(MAIN- CREATE/PROCESS/ADD/(PORTS-VARIABLES -CONSTANTS))*:
  - The user selects 'Create '→'Process' → 'Add'. The user can add ports, variables, constants, generics, packages etc. So if the user selects 'Ports' then is prompted for the 'name' of the port and the user needs to enter the name and press 'carriage return' key and then needs to select a 'data type' from the menu. If the user selects a vector type then the tool prompts for the upper and lower limit. Then the control returns back to

‘Add’ menu. The procedure needs to be repeated for any addition of ports or variable or constants.

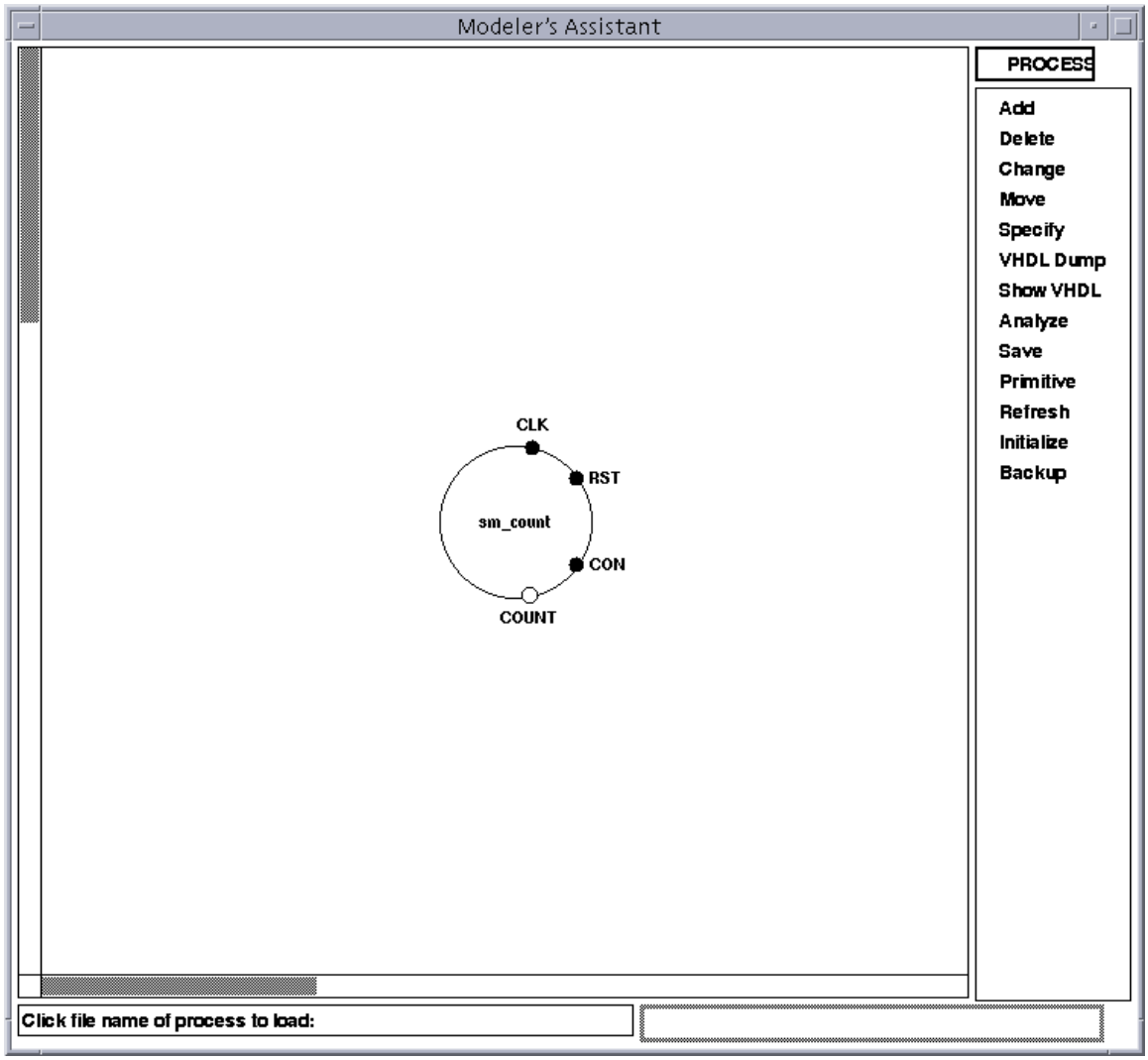

Figure 5.1 The Process Menu

- *Adding Packages(MAIN-CREATE/PROCESS/ADD-PACKAGE)*:
  - The user needs to select the ‘Packages’ and then needs to select one of the packages listed. If the user defined package needs to be added then ‘UserDef’ is to be selected and the name of the package is to be entered when prompted.

- *Adding Generics(MAIN-CREATE/PROCESS/ADD-GENERICS*):
  - When the user selects the ‘Generic’ menu item from the list, the tool prompts for the name of the generic and type of the generic. The user needs to be careful because generics of type ‘time’ are not synthesizable.

- *Adding Sense*(*MAIN-CREATE/PROCESS/ADD-SENSE):*
  - The user can sensitize a port by selecting this menu item. The user then clicks on the port to be sensitized. The port now is represented by a ‘filled circle’ to indicate the fact that the port is sensitized. This port will appear in the sensitivity list of the process.

- *Initialization(MAIN-CREATE/PROCESS-INITIALIZE):*
  - The ports, variables, generics etc can be initialized using this particular menu item.

- *Deletion(MAIN-CREATE/PROCESS-DELETE)*:
  - If the user makes any mistake while in the process of adding or initializing then ‘Delete’ menu needs to be chosen and the appropriate should be deleted. For example, if the user wishes to delete a ’port’ , then the user needs to choose ‘port’ and click on the port to be deleted.

- *Specify(MAIN-CREATE/PROCESS-SPECIFY):*
  - The user selects the ‘Specify’ and then enters the functional parts of the process in the space given. Then needs to press ‘OK’.

- *Save(MAIN-CREATE/PROCESS-SAVE):*
  - The user needs to ‘Save’ the process and then dump the model to generate the VHDL. The user is prompted for the name of the file, carriage return is hit in response if the user wants the name of the VHDL file and the name of the process to be the same. If the user chooses it to be different, then he/she can enter a new name.

- *Analyze(MAIN-CREATE/PROCESS-ANALYZE):*
  - The user needs to choose ‘Analyze’ and needs to enter the name of the module , unit and name of the process to be analyzed. The user can just hit carriage return in response so that the name of the process would be used as default. If the user dumped the VHDL using any other name then he needs to enter that name in response to the name of the file to analyzed.

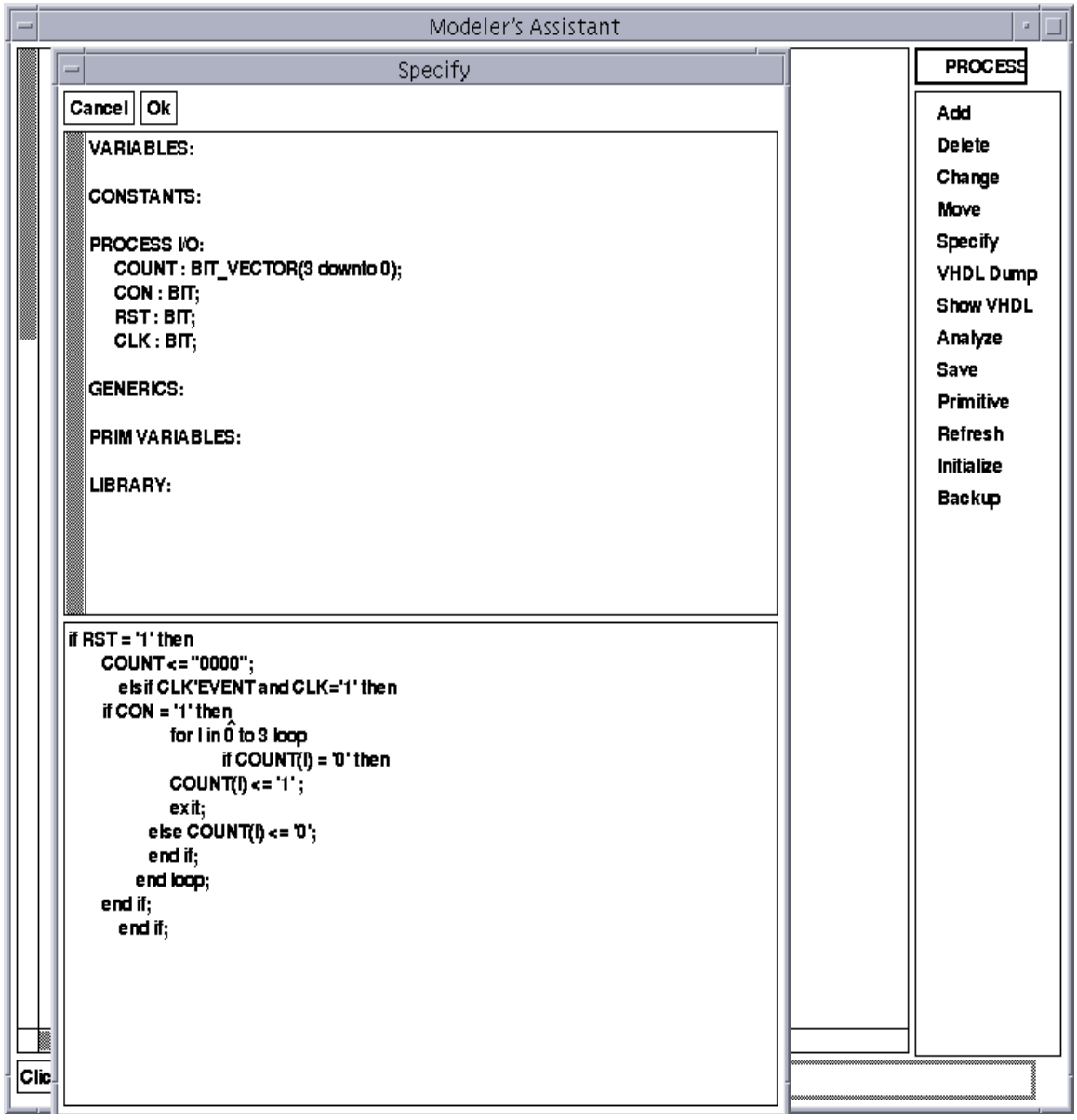

Figure 5.2 The 'Specify' window

Step 2: *(MAIN-CREATE/UNIT)*

Once the user builds all the processes required in the model then, the next step in the development would be to build a ‘unit’ to represent the whole model , that is ,the complete process model graph (PMG). Figure 5.3 shows the ‘Unit’ Menu.

- *Unit(MAIN-CREATE/UNIT):*
  - The user needs to select ‘Unit’ under ‘Create‘ menu and needs to enter the name of the unit. This name is used to name the .vhd file after the user dumps the model.

- *Adding Processes(MAIN-CREATE/UNIT/ADD-PROCESS):*
  - Then the user needs to select the ‘Add’ menu item and proceeds to add the previously built processes to the design space on the screen. The user can place the process anywhere in the design space. Figure 5.4 shows the ‘Add’ menu.

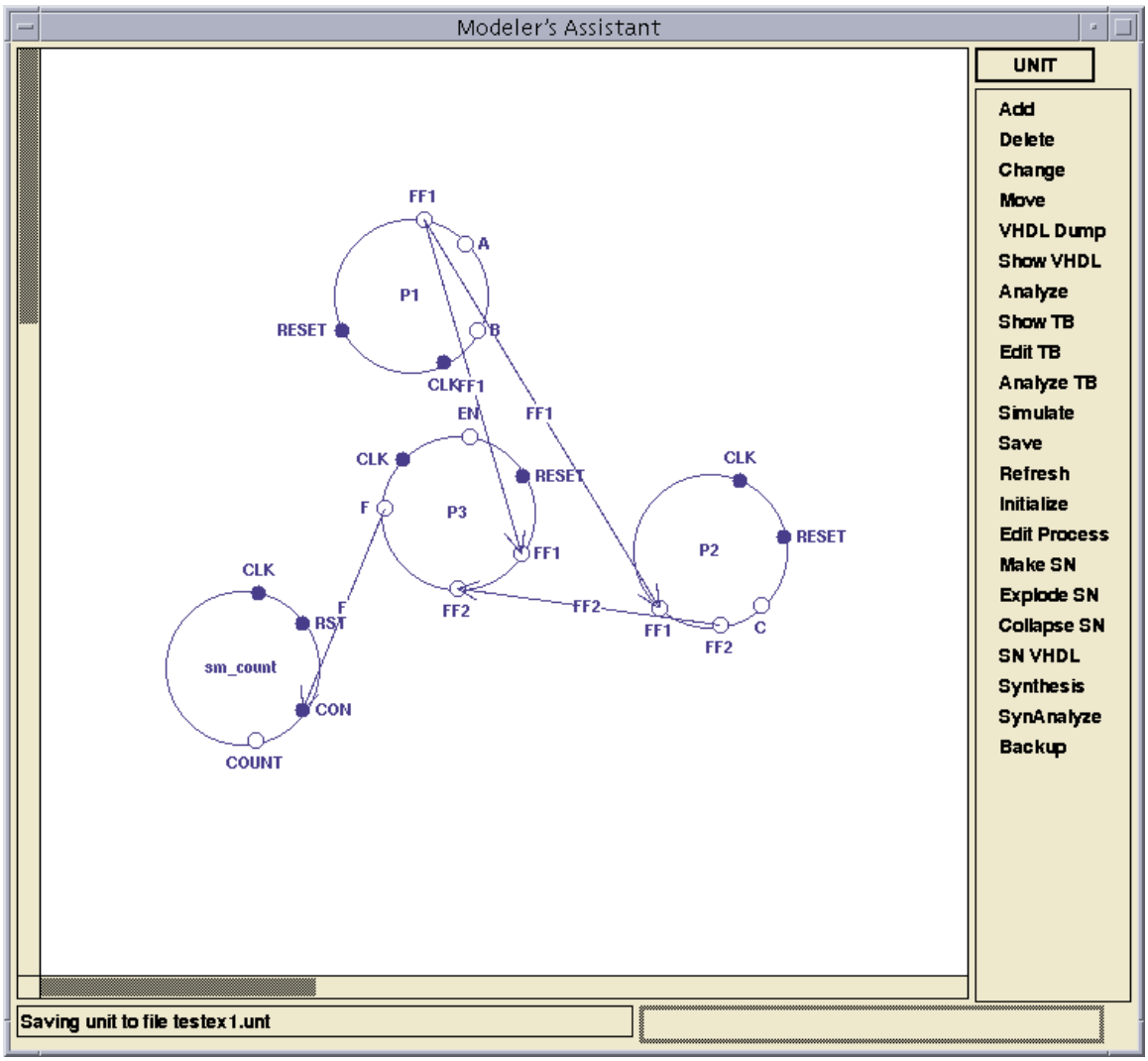

Figure 5.3 The 'Unit' Menu

- *Adding Signals(MAIN-CREATE/UNIT/ADD-SIGNAL):*
  - Once the user adds all the processes in a unit , then he/she would select 'Signal' to add signals which would actually represent the flow of data from one process to the other. The user needs to make sure which process ports are to be connected. If the user tries to connect a port of mode 'in' to any other port of mode 'out' or if the data type of port is not same as the destination port or if user tries to add a signal between the ports bearing

the same mode then the tool beeps and warns the user that a signal cannot be added. Once the user adds the signal , he/she is prompted for the name of the signal. He/She can hit return or specify a name. This name replaces all the corresponding port names wherever they appear and therefore they became the ‘internal signals’. Several signals can have the same source port but more than one signal cannot have the same destination port. Then the user needs to ‘Backup’ to the unit Menu

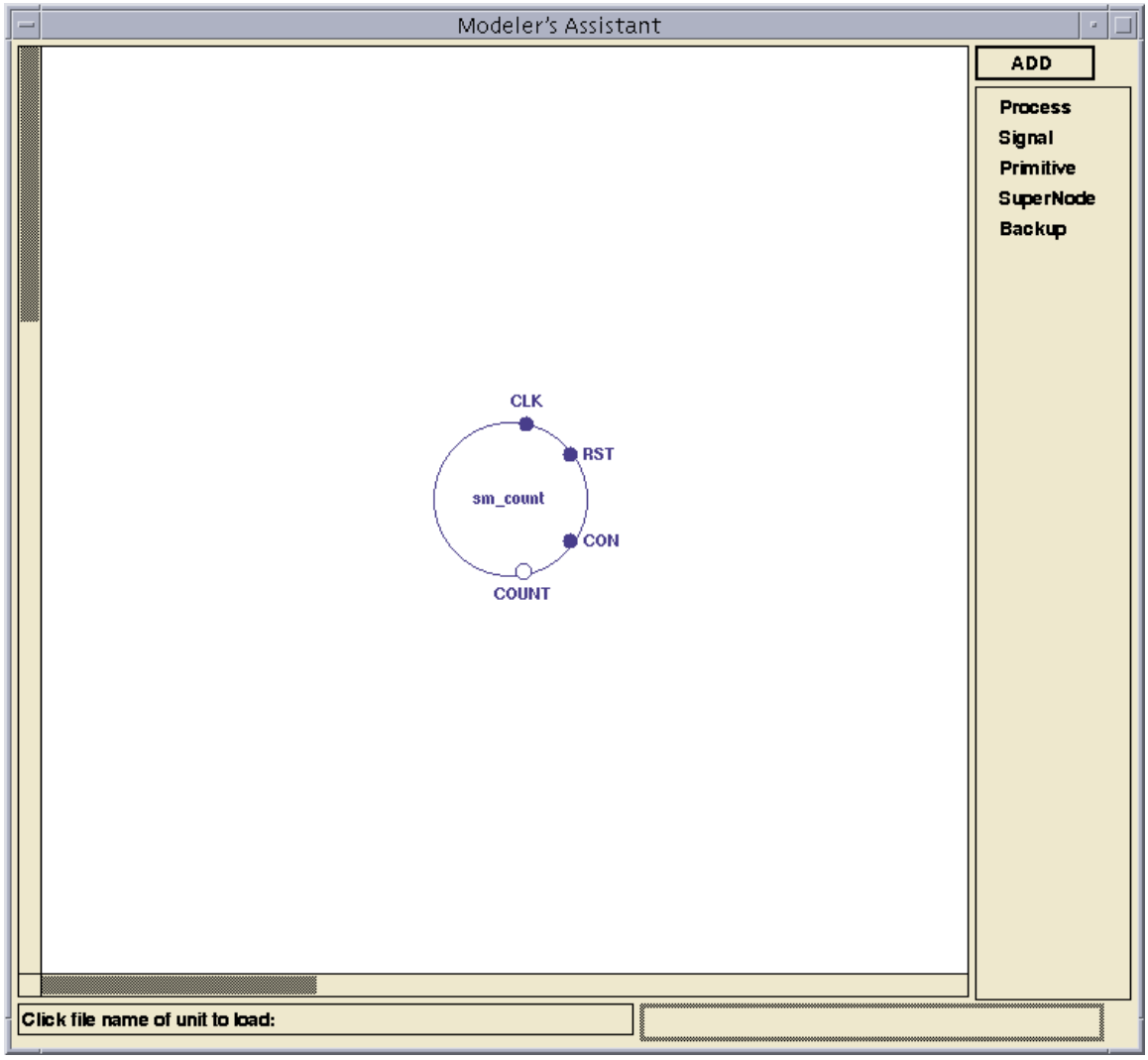

Figure 5.4 The ‘Add’ Menu

- *Generating the VHDL(MAIN-CREATE/UNIT-DUMP VHDL):*
  - The user then ‘dumps’ the model. This would generate VHDL in which the name of the entity is the name of the PMG , that is, the name of the unit entered. The entity ports are those ports which are left ‘unconnected‘ to any signal. If any signal matches another in name and type , it is assumed to be the same signal. No connection is required, that is, no signal is added explicitly to specify this fact. When the user selects the ‘Dump VHDL’ menu item, the user needs to give his/her response regarding if the model is for Validation. Answer ‘n’ at this stage if it is not for validation. Then proceed to view the VHDL generated by selecting the ‘Show VHDL’ menu item. The user is prompted for status lines, if the user chooses to respond with ‘y’ then status lines are shown. This is just for debugging purposes. One can observe that architecture is always named as ‘BEHAVIOR ‘ and all signals are declared inside the architecture and the variables inside the processes. Click on ‘OK’ and save the file by choosing the ‘Save’ menu item.

- *Analyze(MAIN-CREATE/UNIT-ANALYZE):*
  - Then the user chooses the ‘Analyze’ option to analyze the file which would aid the user in knowing the errors committed. The user is then prompted to check if it needs to be done with ‘synthesis switch’. Depending on the response from the user the analysis is done with ‘spc_elab’ or ‘-c’ option. Then the user needs to enter the name of the file to be analyzed. As a default, the user can hit ‘carriage return’. If the user enters any other name when he dumps the VHDL model then, that file name is to be entered whenever the tool prompts for the filename. Else the user can hit ‘carriage return’ and the name of the PMG (given by the user) is used for all the purposes. Then a window opens up which would show the results of the analysis. Any errors reported need to be corrected by going back to the processes by choosing the ‘Edit Processes’ menu.

Step 3: (MAIN-CREATE/UNIT-EDIT TB):

Once the VHDL file generated by tool is verified for the syntax errors, the user then can go in to build the test bench to test the model. The test bench is used to apply the input stimuli and can be therefore used to check the compliance between the functional behavior of the model and the desired behavior. We would consider different types of the test bench. The test bench is saved with _TB.VHD extension except for the one generated by the 'Syncad'.

- *Generating Test Bench(MAIN-CREATE/UNIT-EDIT TB-SHELL):*
  - This would generate a simple test bench with no automatically generated vectors or with no facility to read the vectors from files. This test bench would have entity, architecture, component and configuration declaration. Refer Figure 5.5 to view a sample of a test bench.

  - *No Generated test bench exists or edits occurred, make new[y/n]?:*The user needs to answer 'y' if the test bench has not been generated. This feature proves useful if the user commits a mistake while entering the test vectors manually and realizes it when the test bench is analyzed. The user then needn't regenerate the whole test bench but can answer 'n' to the above prompt and then edit the test bench. But if the changes are made to either the PMG database or the process database, then user needs to regenerate the complete test bench.

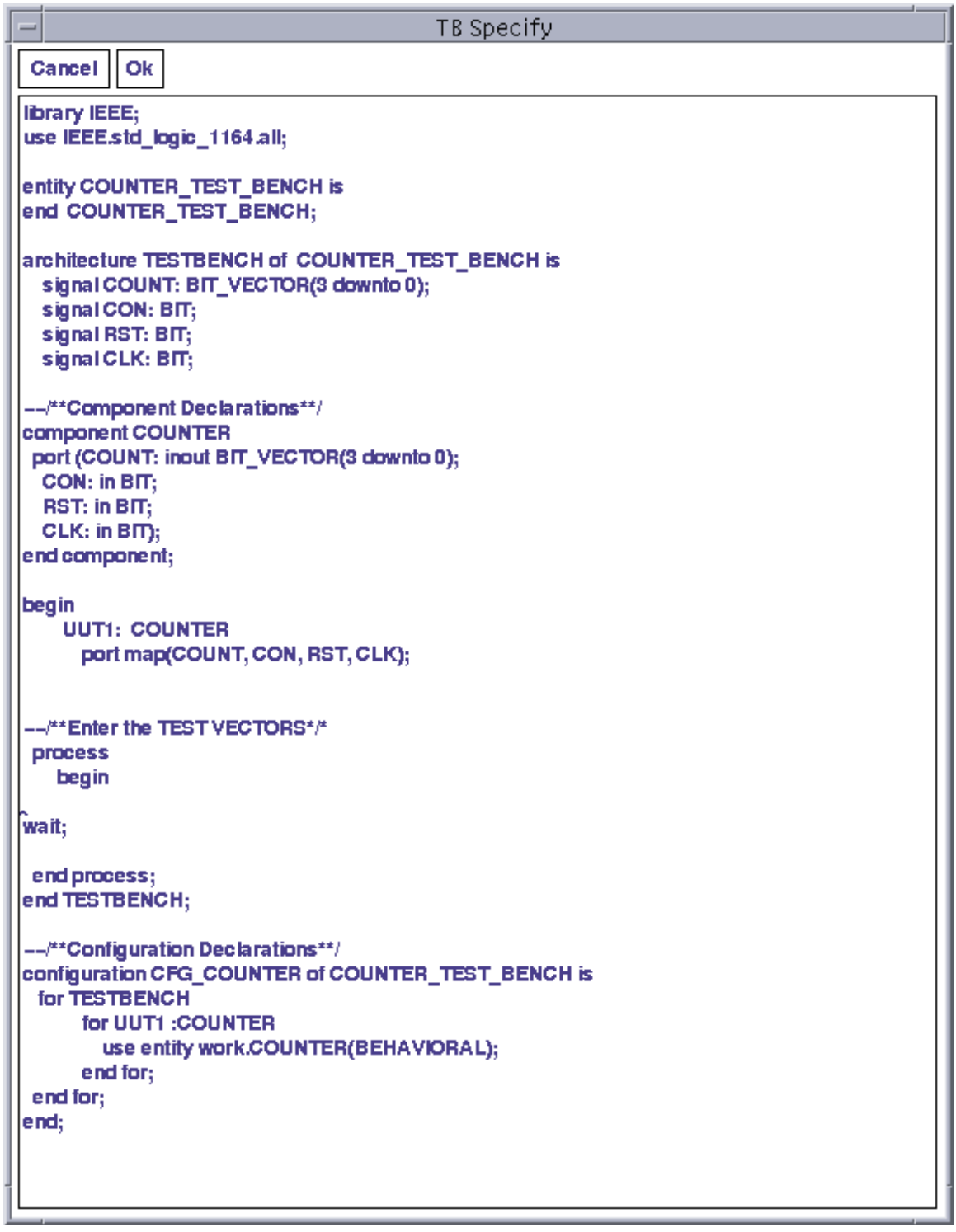

Figure 5.5 The 'Shell' Test Bench

- *Generating test bench for combinational circuits(MAIN-CREATE/UNIT-EDIT TB-CG):*
  - This type of test bench, as mentioned earlier, is quite useful if the user desires to test all possible combinations of a set of inputs. Say, the user is modeling a design which has 4 inputs whose all possible combinations needs to be applied to test the model effectively. Lets call them J, K, S, R. The output of the combination generator is called PG. Therefore for port mapping purposes any of the input can go to any of the PG. Say, the user desires that PG(0) should be mapped to signal 'J', PG(1) to 'K' etc. The user is prompted for following.

    - *Enter length of generic 'N':* The generic 'N' that would represent number of the inputs for which all possible combinations is desired. In case the number of inputs are four, therefore N =4.

    - *Enter value of the delay between changes of combinational generator:* This would represent the delay between changes(from one combination to the other) of the generator which would output all possible combinations of the input of length 'N'. Enter just the number. The default time unit is "ns".

    - *Enter which bit of PG should be mapped to <signal name>, else enter 'n' :* This would require the user to enter which bit of PG should go to which particular input signal. Say, when the signal name is 'J' ,the user is supposed to enter PG(0), if the signal name is K, then needs to enter PG(1) etc. Then that bit of the PG in the port map declaration replaces this signal. If the signal name is not required, then enter 'n'/'N', implying the user doesn't like to map that signal to any of the bits of the output of the combinational generator. This is prompted till all the signals in the PMG are covered.

- Then the tool would generate the test bench with the inputs mapped to the output of the combinational generator. Press ‘OK’ on the test bench window to save the file. The file is saved with ‘_TB.VHD’ extension.

- *CG + OSC(MAIN-CREATE/UNIT-EDIT TB-CG + OSC):*
  - This type is desired for the sequential circuits involving the clock. The ‘OSC’ primitive generates the clock of desired frequency. The user needs to port map his CLOCK signal to the output of the oscillator, called OSC. The user therefore needs to enter the name of the clock, the ‘hitime’ (time for which the clock remains at ‘1’), the ‘lotime’ (time for which the clock remains at ‘0’). The relative ratio of ‘hitime’ and ‘lotime’ gives the duty cycle of the clock. If modeling a JKFF flip-flop, then the model can be tested for all possible combinations of J, K, S, R using the combinational generator

    - *Enter the name of the clock signal:* The user is required to enter the name of the clock signal used in the design.

    - *Enter the value of generic HITIME:* The user needs to enter the value of the HITIME. Enter the number with no time unit. The default used is “ns”. Lets say the user requires a clock of period 100 ns , with a duty cycle of 50% , the HITIME is 50 ns.

    - *Enter the value of generic LOTIME:* The user enters the value of the ‘lotime’ required. If the above case of 100 ns period and 50% duty cycle is considered, then the LOTIME is 50 ns. The user enters only ‘50’ and not ’50 ns’.

    - *Enter the delay between changes of the combinational generator:* The user is required to enter the delay between changes *(*generic PER). Enter just the number with no time units. Make sure that this delay must at least be equal to clock period

time else the model doesn't guarantee the desired functionality because of the timing problems. Study the two primitives 'pulse.vhd' and 'osc.vhd' for more details.

- *Enter which PG should be mapped to <signal name>, enter 'n' else:* Enter which bit of the PG (recall the output of the combinational generator is PG) maps to this particular signal. Then this signal is replaced by that bit of the PG in the port map declaration. If that particular signal is not to be mapped to any bit of the PG, then enter 'n'/'N'. This signal wouldn't map to combinational generator output. But would retain the same name in the port map. This would be prompted for all the signals present in the model.

- Once the user enters the required information, the test bench window pops up and the user needs to press 'OK' so that the file is saved. Figure 5.6a and 5.6b show the test bench.

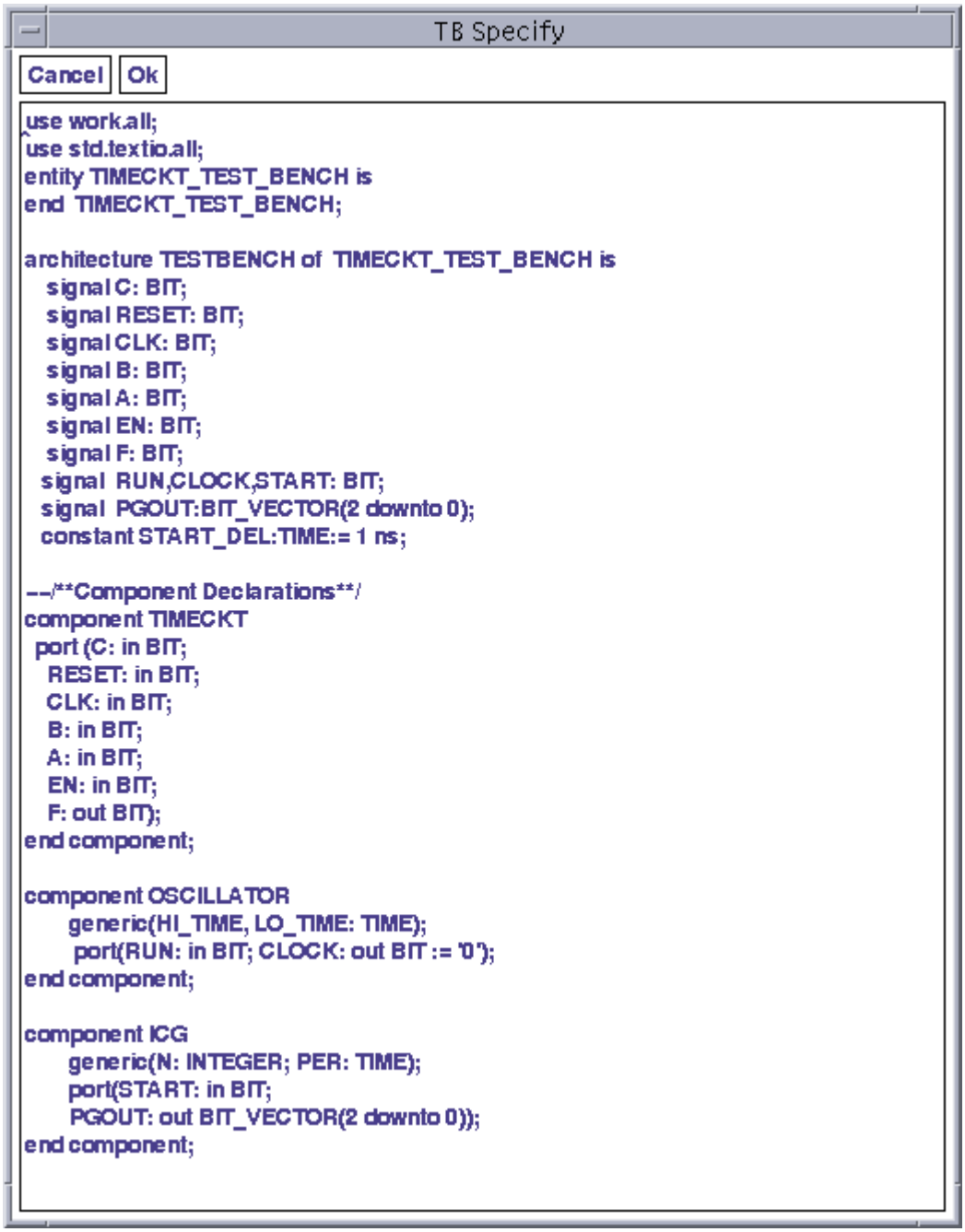

Figure 5.6a The 'CG + OSC' Test Bench

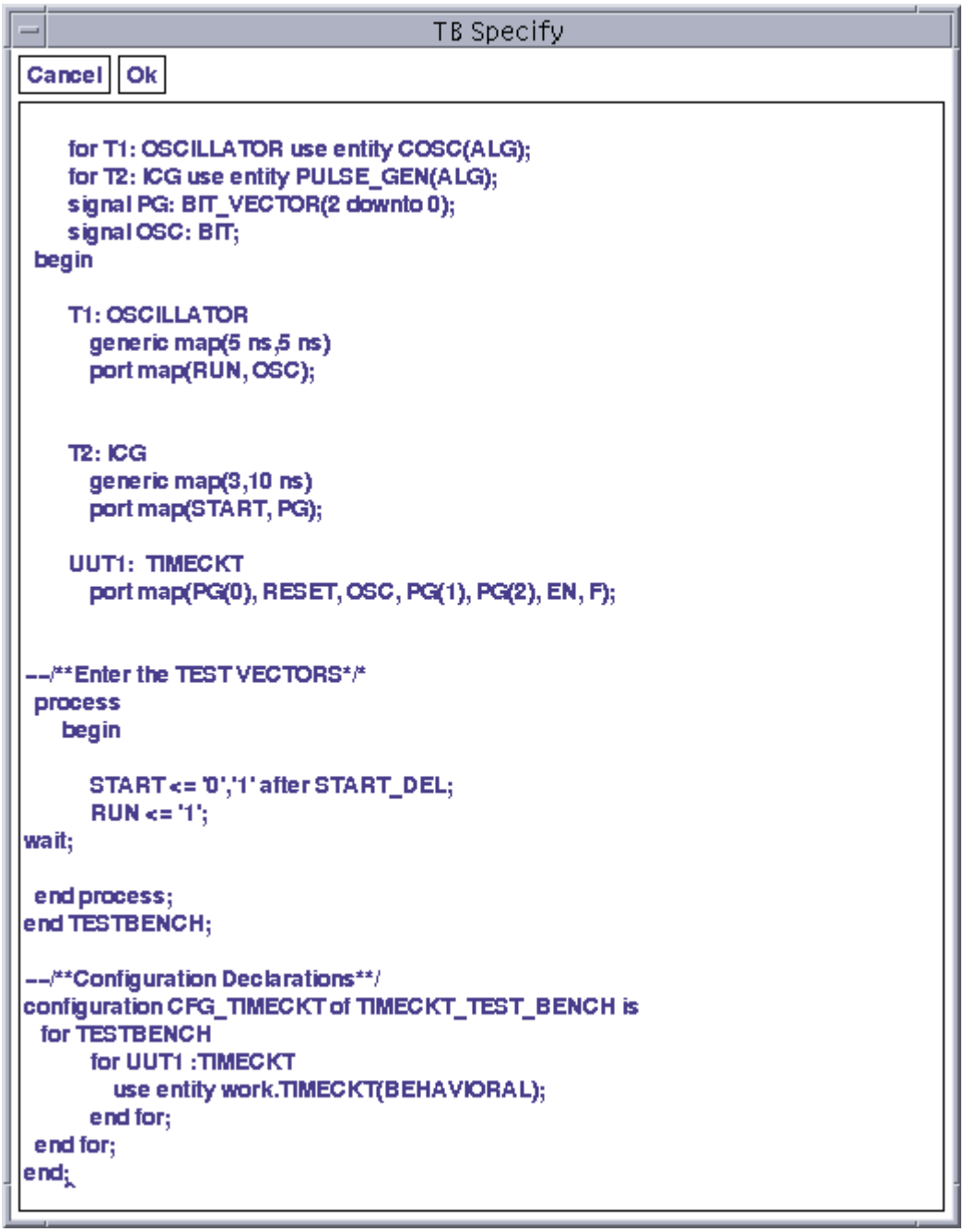

Figure 5.6b Continuation of ‘CG + OSC’ Test Bench

- *Generating test bench for Validation(MAIN-CREATE/UNIT-EDIT TB-VALIDATION):*
  - This is mainly useful when the user desires to compare the synthesized and the behavioral model for select few signals. The user can simulate the model to study the delay differences involved in the synthesized and behavioral model. The delays experienced are gate delays. The synthesized model is a structural model wherein the component is described in terms of interconnection of primitives[1].The user is prompted for the following. Figure 5.7a and 5.7b show this type of test bench.

  - *Enter if the signal <signal name> is to be validated?:* The user is prompted with the name of the signal and is expected to answer in 'y'/'Y' or 'n'/'N'. If the user enters 'y'/'Y', then the signal in the behavioral model is validated against its corresponding in the synthesized model (the signal in the synthesized is called "<signal name>_S"). The user needn't concern with the name and just enter the response with regards to that signal.

  - *Enter the delay for' CHECK_EN' signal.* The CHECK_EN is used to check the validation irrespective of its failing anywhere till the simulation is complete. This would help to continue the validation without stopping so that the user can know where the behavioral and synthesized models do not agree. The delay is for the validation process to start up. The signal is assigned '1' (Refer figure 5.9a and 5.9b) after this particular amount of delay and is asserted till the end of the simulation cycle.

  - *Is the circuit sequential (y) /combinational(n)?:* The user needs to enter 'y' as his response if the model under development is sequential in nature. Else needs to responds with 'n'.

  - *Enter the name of the CLK/STRB:* This would be prompted if the circuit were sequential in nature. The user needs to enter the name of the CLOCK or STROBE signal.

- *Enter if CLK/STRB is Rising Edge:* The user is required to enter with ‘y’ if CLOCK or the STROBE is rising edge or ‘n’ if it is a falling edge.

- *Enter if the signal <signal name> is to be validated?:* Enter ‘y’ if the signal is to be validated else respond with ‘n’.

- The test bench for the validation purpose is generated and a window is popped up which shows the test bench. Press ‘OK’ to save the file. The file is saved with _TB.VHD extension.

- *Generating test bench with Syncad(MAIN-CREATE/UNIT-EDIT TB-SYNCAD):*
  - This test bench is quite well suited if the user doesn’t want to manually enter the test vectors but would like to draw language independent waveforms. When the user selects this menu item, it opens up the Synapticad software called “tbench” in which the user can draw signals.

    - The user can choose ‘Add Signal’ and use the mouse pointer to draw the waveforms. If the user double clicks the signal name then a dialog box opens up in which the user can make his selection regarding the type, name, etc of the signal. One should make sure that the signal name matches the name in the PMG.

    - The user can add ‘CLOCK’ by selecting ‘Add Clock’ and the required frequency.

- Once the user completes the waveforms corresponding to various signals ,he would have to choose ‘Export’ and drag the mouse pointer to ‘Export Signal As’ and then the file is saved with ‘VHDL wait’ or ‘VHDL Transport’ format under ‘List Files of Type’. Save the file with a name. Choose ‘Exit’ from the file menu to quit from the software. Refer figure 5.8a and 5.8b for test bench with wait statements. Refer 5.9a and 5.9b for the test bench with transport statements.

- Once this is done, the user can choose ‘Show TB’ and the user is prompted for test bench with automatically generated vectors. Choose ‘y’ and then is prompted name of the file saved .The user needs to enter the name of the file in which he exported the signals using the ‘Syncad’ software. The test bench opens up with the vectors included.

- Remember that the user first must specify the required type of test bench, like Shell or CG or File IO etc and then use ‘Syncad’ to draw the waveforms. The vectors from the saved file are added to the mostly recently saved test bench. This new test bench is saved with “_tbench.vhd” extension. Therefore the user can still use _TB.VHD if chooses to use the test bench without these automatically added generated vectors.

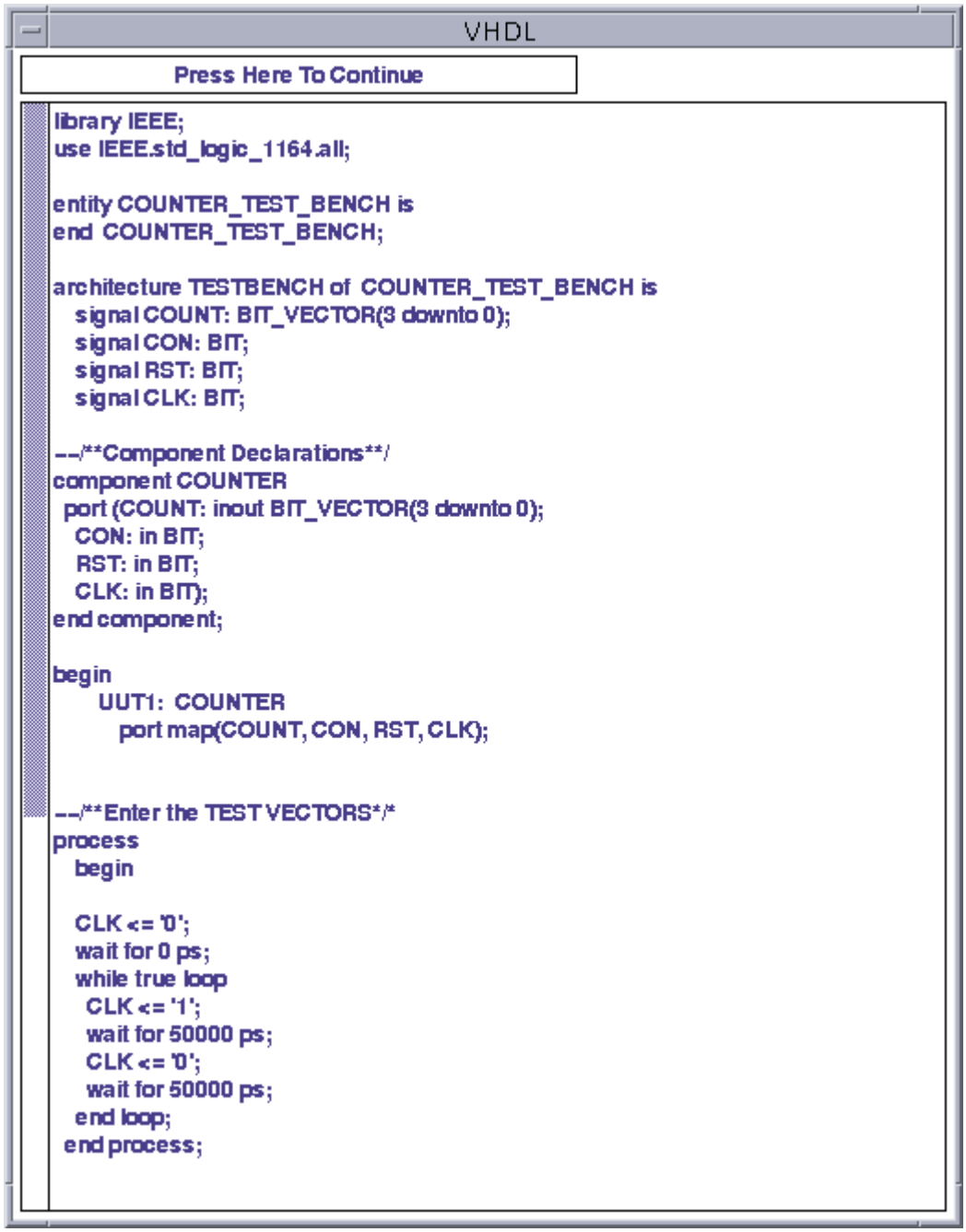

Figure5.7a The Test Bench with 'Wait' statements

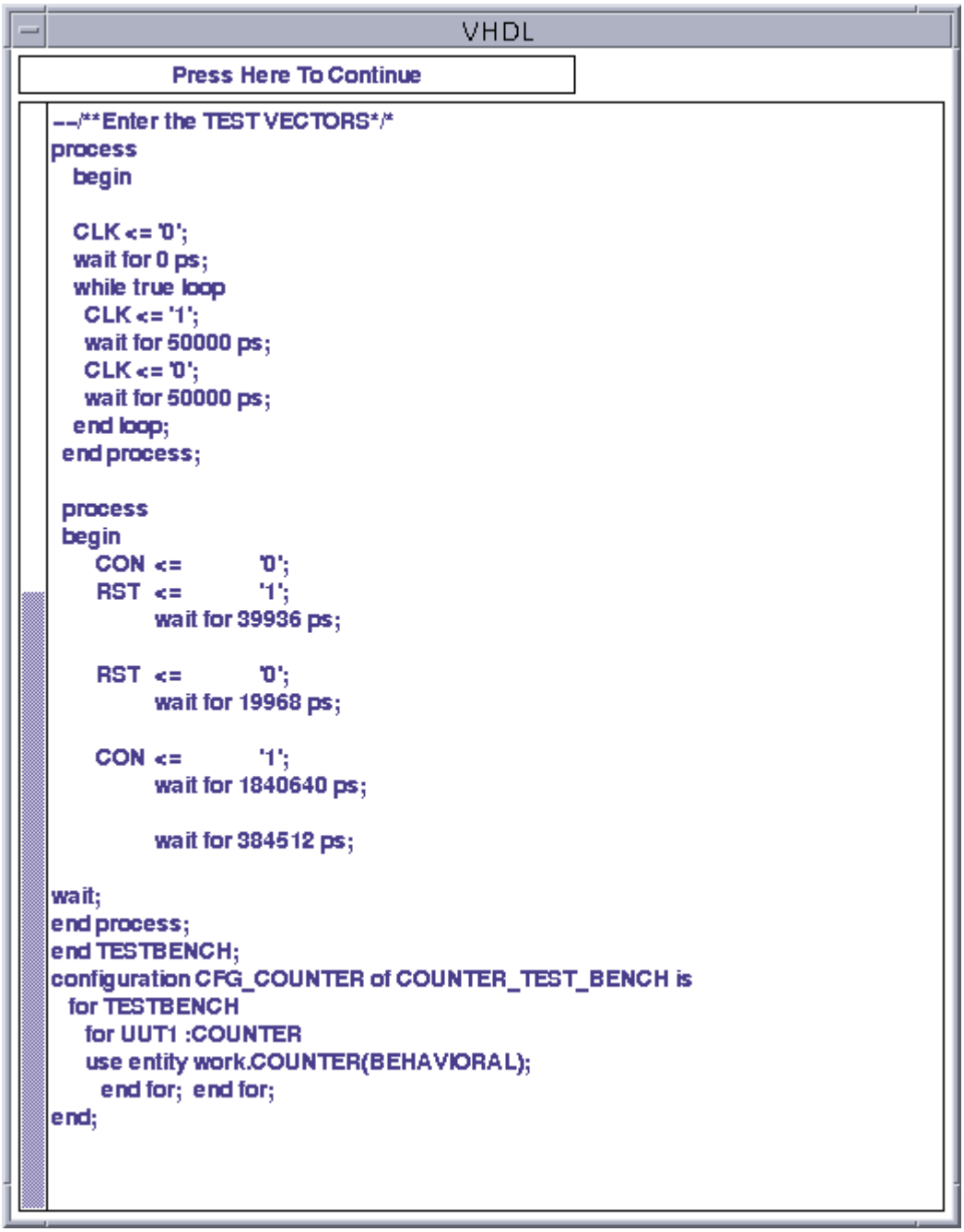

Figure 5.7b Continuation of the Test Bench with 'Wait' Statements

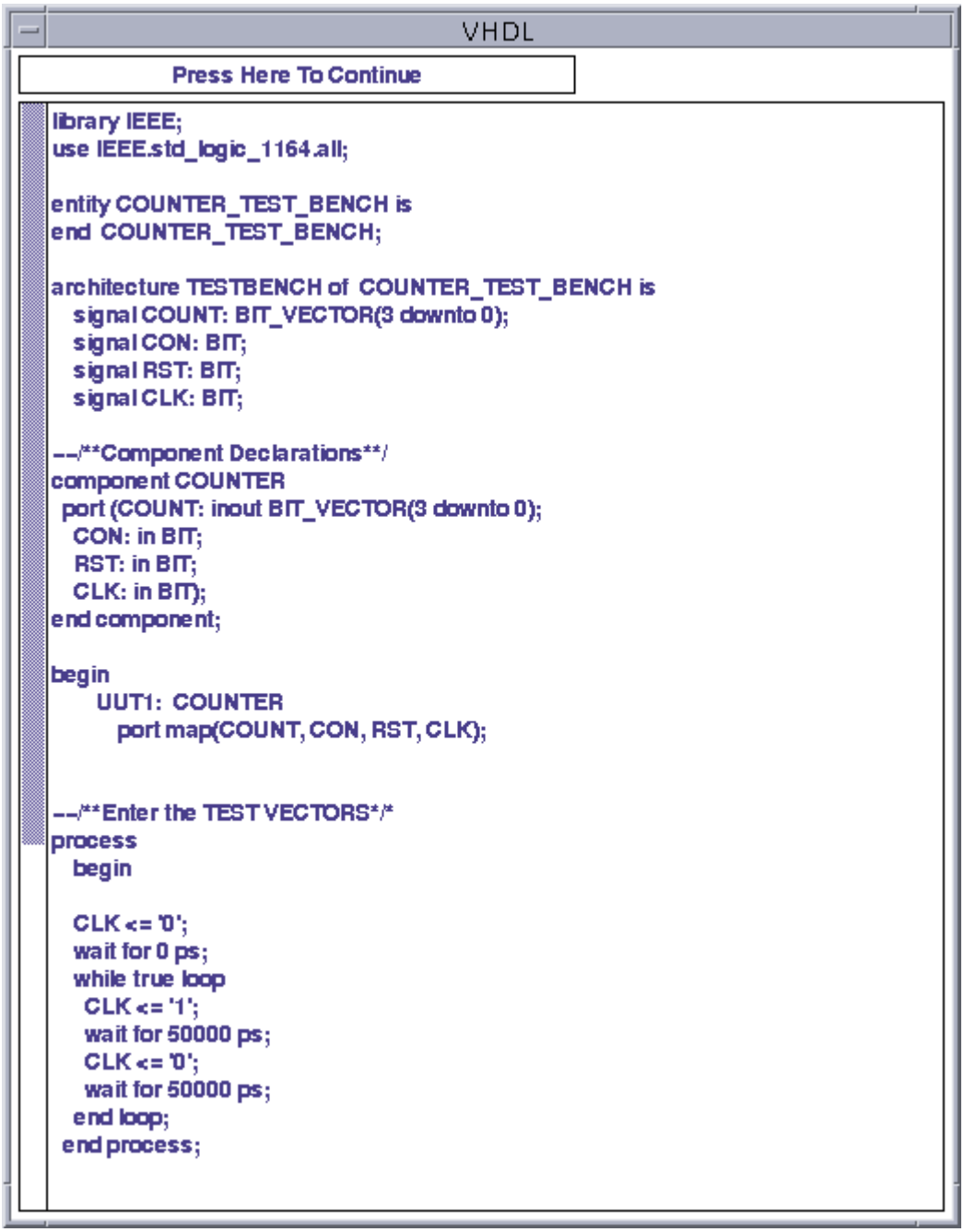

Figure 5.8a: The Test Bench with 'Transport' Statements

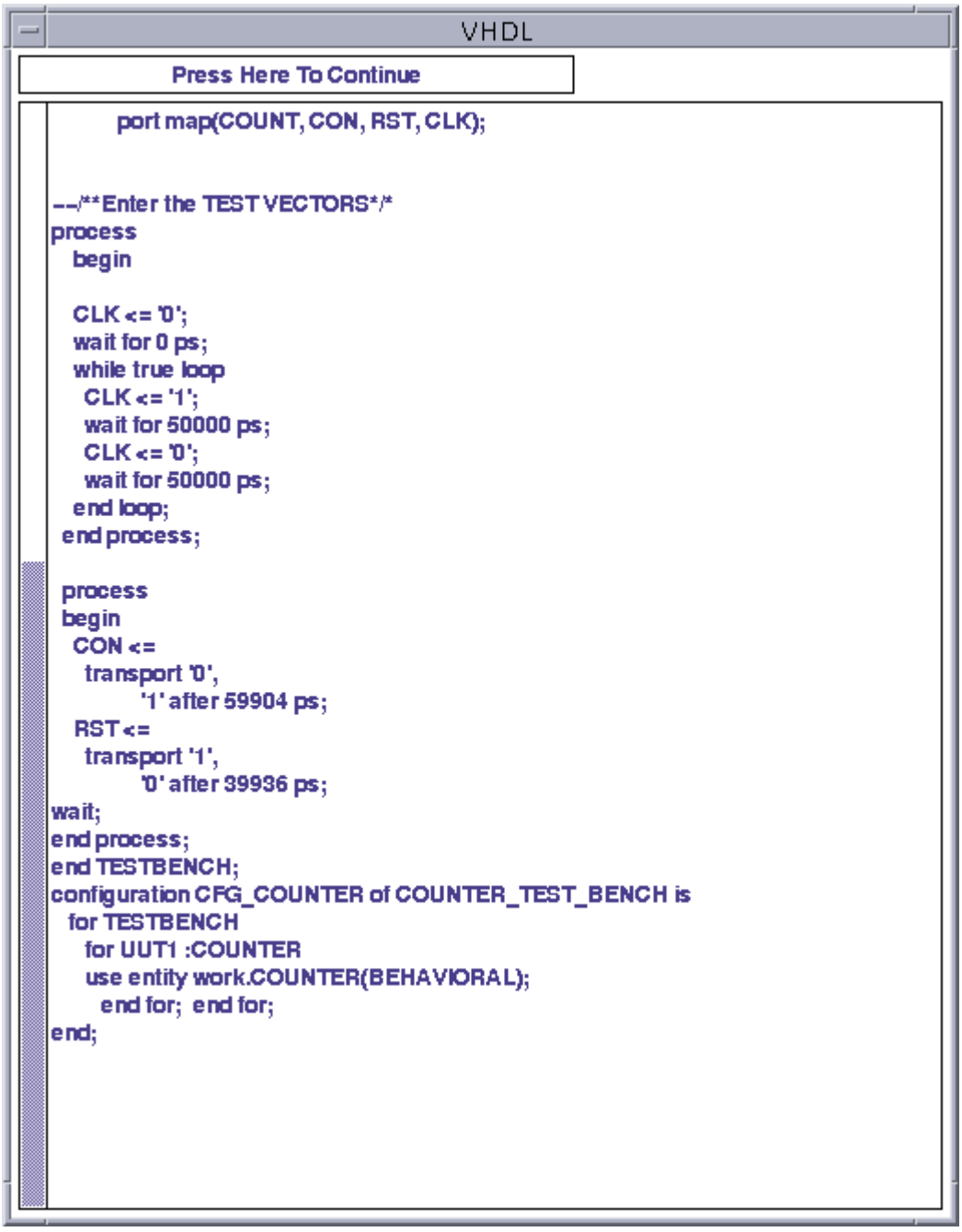

Figure 5.8b Continuation of the Test Bench with 'Transport' Statements

- *Generating the test bench in which test vectors are read from FILE(MAIN-CREATE/UNIT-EDIT TB-FILEIO):*

  This type of test bench is used when the user desires to read the test vectors from a file instead of manually entering them. This type of test bench uses the 'Text IO' capability of VHDL language and is useful when memories are involved in the model.

  - *Enter the name of the file:* The user is prompted for the name of the file which contains the test vectors. The user should enter the name including the extension (say,'.txt').

  - *Enter the number of test vectors to be read:* The tool prompts the user for number of vectors to be read from the file. The user is required to enter the number, say, like 2 or 3 depending on the number of variables to be read.

  - *Is the variable vector type?make [y/n]:* The user is required to enter 'y' if the variable is vector type. Else should enter 'n'/'N'.

  - *Enter the limit m:* This is prompted only if the user answers 'y' for vector type. The user needs to enter the upper limit or lower limit depending on whether 'to' or 'downto' is required.

  - *Enter the limit n:* Then the user is prompted for the other limit of the variable size. If the number 'm' is greater than 'n', it is printed as "m downto n". Else is printed as "n to m".

  - *Assign variable V to signal?:* The user is required to assign the variables from the file to signals in the model. Therefore enter the name of the signal to which V1 or

V2 (consider case of two vectors to be read) is to be assigned. Do enter the name so that name and type would match that particular ‘V’. All variable are referred to as V1, V2 etc.

- Once the user enters all the information, the test bench window pops up and would show the generated test bench file. The file is saved when the user presses ‘OK’ in the window.

- *Generating test bench with vectors read from File IO and with clock generation(MAIN-CREATE/UNIT-EDIT TB-FileIO + OSC):*

  - This type of testbench is used when the modeler is designing a clocked sequential circuit where the test vectors can be read from a file and the CLOCK is generated by using the oscillator primitive.

  - *Enter the name of clock signal:* The user needs to enter the name of the clock. This signal name is replaced by “OSC” in the port map declaration. Thus the output of the oscillator would drive the clock in the model.

  - *Enter the HITIME:* Enter the time for which the clock edge is at logic’1’. The user is expected to enter the number and not the time unit. The default time unit used is “ns”.

- *Enter the LOTIME:* The user is expected to enter the time for which the clock is at '0' or is low. Note the relative magnitude of the 'HITIME' and 'LOTIME' determines the duty cycle of the clock.

- Follow the steps illustrated in the FileIO section after this step.

- Once all information is being given the test bench window pops up and the generated test bench is shown. To save the file press 'OK'. This file is saved with '_TB.VHD' extension.

- **Step 4** :Viewing the test bench(MAIN-CREATE/UNIT-SHOW TB):
  - Once the test bench has been generated the next step would be to view the generated test bench. The following information is required to proceed.

    - *Is the test bench with generated test vectors:* If the user chose 'Syncad' software to supply the test vectors and desires to view the complete test bench with the generated vectors included, he /she needs to respond with 'y'. With 'y' as a response the test bench shown is the one with generated test vectors from Syncad software. If user responds with 'n', the shown test bench is one with manually entered test vectors or the one in which the test vectors are read from file depending on the most recently chosen test bench.

- *Enter the name of the file saved in Syncad*: If the user responds with ‘y’ for the above step, then the next response from the user needs to be the file name where he/she exported the signals to. Then the window pops up to show the test bench.

The figure 5.9a and 5.9b show the test bench window. The test bench chosen is ‘Validation’. No test vectors are added to the test bench.

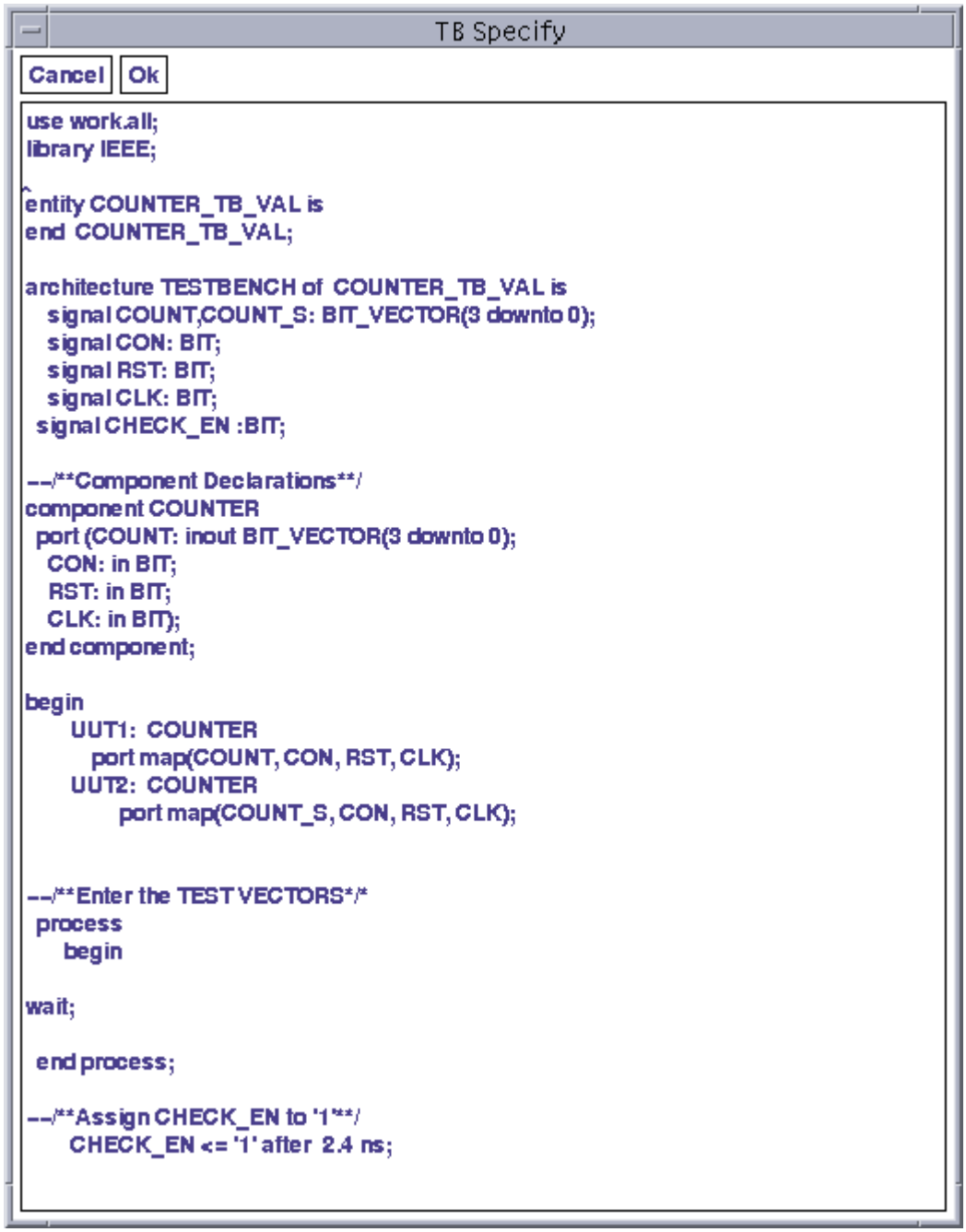

Figure 5.9a The 'Validation' Test Bench

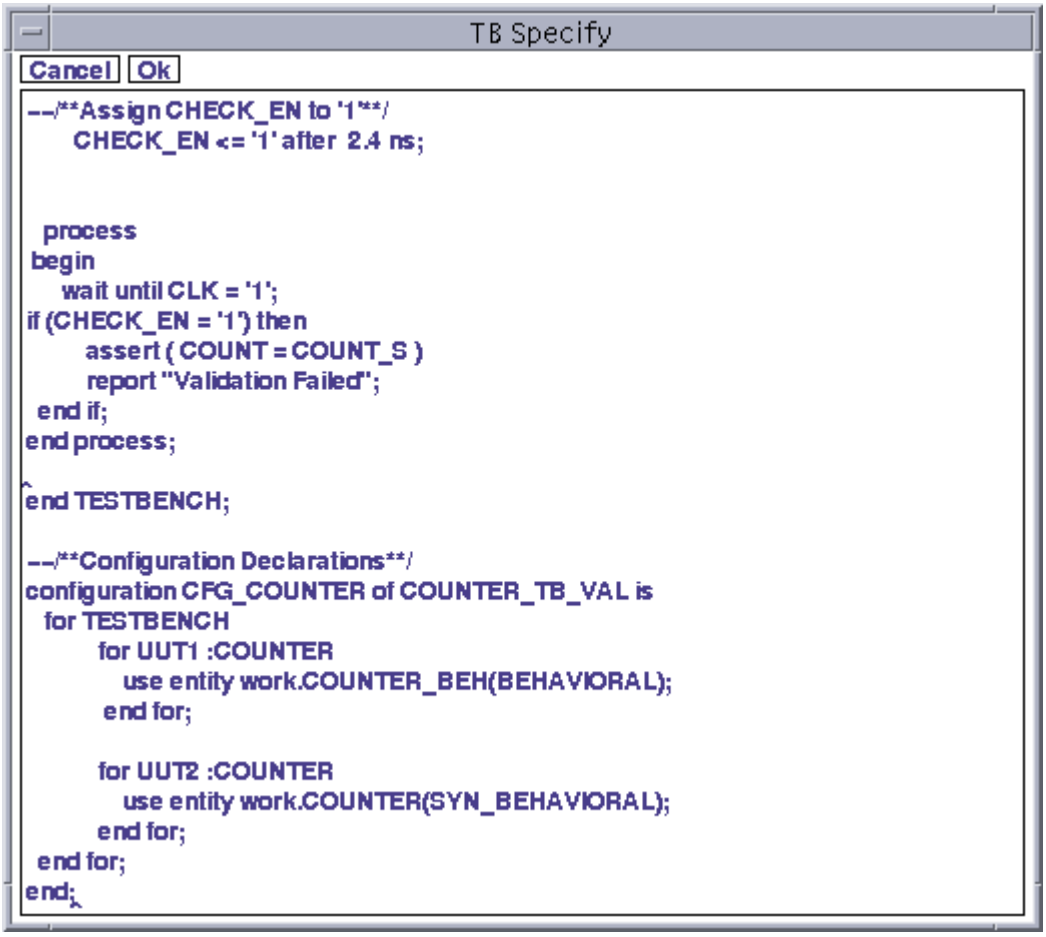

Figure 5.9b Continuation of 'Validation' Test Bench

- **Step 5:** *Analyzing(MAIN-CREATE/UNIT-ANALYZE TB):*

  - Once the test bench has been generated, the next step in the process of development is analyzing the generated test bench.

  - *Analyze the test bench for the synthesized circuit:* If the test bench is used for the post synthesis process the user should respond with 'y' and the analyzing option is "vhdlan –t ps –xsim <filename>".Else if the user responds with 'n' the analyzing option is "vhdlan <filename>".

- *Analyze with automatic generated test vectors?:* If the user responds with ‘y’ then the test bench analyzed would be the one with generated test vectors added to the already existing test bench. That is the test bench with ‘_tbench.vhd’ extension. If the user chooses ‘n’ then the test bench is one with manually entered or test vectors read from file.

- *Enter the test bench name*: The user hits ‘carriage return’ response. The tool will take care to see that the most recently created test bench is analyzed.

- **Step 6:** *Simulation(MAIN-CREATE/UNIT-SIMULATE):*
  - The next logical step in the process is simulation to verify the functional behavior of the circuit.

    - Choose the ‘simulate’ menu item in the Unit Menu and a window of ‘Simulator arguments’ opens up. Choose the design and the time units and press ‘OK’. All other setting are the default settings.

    - The graphical debugger window pops up. Trace the required signals and simulate the design. If the design doesn’t work as desired, then close the simulation window and edit the processes or the test bench to achieve the desired behavior.

- **Step 7:** *Synthesis(MAIN-CREATE/UNIT-SYNTHESIS):*
  - Once the simulation results match the desired or expected behavior then the next step would be to synthesize. Figure 5.8 shows the 'Synthesis' window. A good practice is to analyze the VHDL source file with synthesis switch as it would indicate whether the design is synthesizable. Do not use generics of type 'time', as they wouldn't be synthesizable.

    - The user needs to create a 'design analyzer script file' with '.scr' extension to synthesize the design. the format for this script file is shown in Figure 4.7. Refer [4] for details about the script file.

    - *Enter the name of the script file:* The user is expected to enter the name of the file. No file extension is required. The tool concatenates the file name with '.scr' extension.

    - The tool opens the 'dc_shell' window and the synthesis would be performed. The synopsys libraries are used in this process. The results are stored in a file with '.rep' and '.vhd' extensions. The '.rep' files contain the reports of the simulation. The report includes area or time or power etc parameters as specified in the script file.
      The '.vhd' file is the synthesized model that is structural in nature.

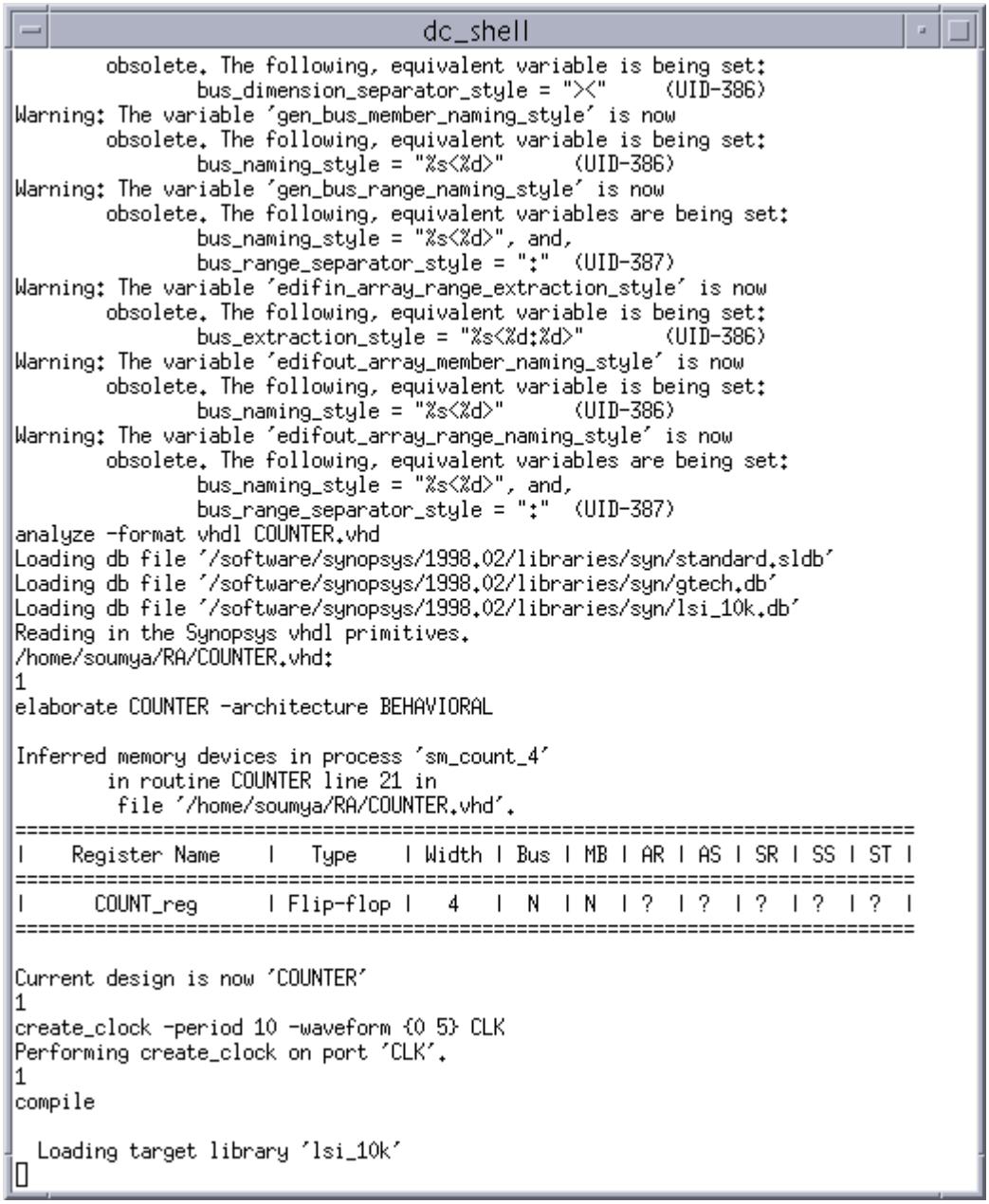

Figure 5.10 The 'Synthesis' Window

- **Step 8:***Analyzing Synthesis file(MAIN-CREATE/UNIT-SYNANALYZE):*
  - The next step would be to analyze the synthesis file in order to simulate and compare its behavior to the behavioral model.

    - *Enter the name of the synthesized file to be analyzed*: The user needs to enter the file name he chose in script file to represent the synthesized '.vhd' file.

    - The analyzer would finish analyzing the file. Once this is done, press 'DONE: Press here to continue'.

- **Step 9**:Analyze the test bench for the synthesized model and then simulate the model for verifying the behavior.

- **Step 10:** If the user chooses to validate synthesized model with behavioral model ,then
  - Choose 'Dump VHDL' to dump the model and respond with 'y' for 'Is the VHDL for validation'.
  - Analyze this file choosing 'Analyze' in the unit menu and respond with 'n' for synthesis switch option.

  - Then create the test bench for validation by choosing 'EDIT TB' and then 'Validation'.

  - Enter the signal names and other information and generate the test bench file.

- Analyze the test bench by selecting the ‘Analyze TB’ option. Respond with ‘y’ for the ‘Analyze test bench for the synthesized circuit’. The user can choose to answer ‘y’/’n’ when the tool prompts ‘Analyze the test bench with automatically generated vectors’ depending on whether the test vectors were added manually or entered automatically.

- Then when it prompts for ‘Enter the name of the test bench’, just hit ‘carriage return’.

- Once the analyzing of the file is done with no errors reported, proceed to simulation.

- Check the simulation and analyze the timing issues involved in both behavioral and the synthesized circuit.

- **Step 11**:DONE!!!

Chapter 6
Future Work

The Modeler's Assistant can be enhanced by adding few more features and by some minor changes to the existing features.

As per the current version the user is required to write the synthesis script file. Therefore would involve some effort on part of the user . Future version may have a menu list from where the user can choose the type of script file and wouldn't have to spend time writing one.

Right now the tool takes in graphical input and produces the VHDL code, the future version can do the vice versa. Take the VHDL code and use some sort of parsing techniques and then generate the process model graph (PMG) as the result.

In this current version, the test benches 'File IO', ' File IO + OSC', 'CG' and "CG +OSC' are available for the data type 'BIT' and 'BITVECTOR'. This can be extended to any data type used.

The data structure for Modeler's Assistant is a static linked list structure wherein the size of the linked list is fixed, efforts may be taken to make this dynamic structure. That would improve the memory efficiency. Thus less memory would be available for small circuits and large amount can be used for larger circuits.

This tool can be a language independent tool and the user can choose to generate a VHDL or Verilog code. The current version is limited to VHDL only.
This would be a major step to improve the usefulness of the tool.

This version does have a 'help' section. Such a section would make it more user friendly.

Bibliography

1. James R. Armstrong, F. Gail Gray, “Structured Logic Design With VHDL”, Prentice Hall, New Jersey, 1989.

2. David M. Dailey, “Integration of VHDL Simulation and Test Verification Into a Process Model Graph Design Environment”, *Master’s Thesis*, Virginia Polytechnic and State University, June 1994.

3. Debugging Tools, Sun Source Browser Reference, Sun Microsystems, Inc.

4. Synopsys Manual, Synopsys, Inc.

5. O’Reilly & Associates,” The Definitive Guides to the X Window System”, Volumes 1- 6, O’Reilly & Associates, Inc.

6. Brian W. Kernigham, Rob Pike, “The UNIX programming Environment”, Prentice Hall, 1984.

7. Byron S.Gottfried, “Theory and Problems of Programming in ‘C’”, Schaum’s Outline Series, Tata McGraw-Hill, 1994.

8. Balraj Singh., “Parameterized CAD Tool for VHDL Model Development with X windows”, *Master’s Thesis*, Virginia Polytechnic and State University.

9. David Burnette, “A Graphical Representation for VHDL Models”, *Master’s Thesis*, Virginia Polytechnic and State University.

10. Balraj Singh, J.Wicks, P.Wright and J.R. Armstrong, ”The Modeler’s Assistant: A CAD Tool for Behavioral Model development,”,CHDL’93.

11. Oliver Jones, ”Introduction to X window System”, Prentice Hall, 1989.

12. Unix Press, ”Programming With Unix System Calls”, Unix Press, 1992.

13. Robert W.Scheifler, Ron Newman, James Gettys,” X Window System-C Library and Protocol Reference”, Digital Press, 1988.

14. Libes, ”Obfuscated C and Other Mysteries”, John Wiley & Sons, Inc., 1993.

APPENDIX A
FOR USERS

REQUIRED SETUP:

1. The users who desire to work on Modeler's Assistant needs to download 'modas' executable into his current working directory. Or download it to directory accessible to every user and add the path in the '.cshrc'. Then create a new directory 'MY_MODAS' and open the executable there. Type in the following near the terminal prompt.
   - mkdir MY_MODAS
   - cd MY_MODAS
   - modas

2. This opens up the Modeler's Assistant window on to the screen. The window can be adjusted for the size and location like any other window.

3. A Few parameters and environment have to be setup in the '.cshrc' of the user. If any of the following are in different directories, then make necessary changes to the following settings. The 'Testbencher Pro ' is the Synapticad software. So if the software is not available, make sure that the software is installed. Others are available as a part of Synopsys software.
   - setenv SYNOPSYS /software/synopsys/1998.02
   - source $SYNOPSYS /admin/install/sim/environ.csh

- setenv VCADPRIM /usr2/cad/modas/primlib
- setenv VCADANALYZER /software/synopsys/1998.02/sparcOS5/sim/bin/vhdlan
- setenv VCADSIMULATE /software/synopsys/1998.02/sparcOS5/sim/bin/vhdldbx
- setenv VCADSHELL /software/synopsys/1998.02/sparcOS5/syn/bin/dc_shell
- setenv VCADTBENCH /project/tpals3/VHDL/tbench/bin/tbench
- setenv WAVEHOME /project/tpals3/VHDL/tbench
- source /project/tpals3/VHDL/tbench/setup.csh

4. Copy the following files to the directory MY_MODAS.
   - VHDLCAD.vhd
   - USER_TYPES.vhd
   - mvhd(executable)
   - pulse.vhd
   - osc.vhd

   The user is required to analyze VHDLCAD.vhd , USER_TYPES.vhd, pulse.vhd and osc.vhd.

5. The ‘X’ resource data base is setup as a default when the user opens the ‘MODAS’.

6. Make sure the VHDL environment is setup before the user attempts to use this software. Else would give the following error.
   snps_setup fatal error: (Severity SNPS SETUP USER FATAL) Environment
   variable $ARCH not set

7. Once the user takes care to setup the environment, he/she can use the software with no problems.

APPENDIX B

For Future Developers

This particular section is for the future developers of this tool. This tool has been built with linked list as its data structure. The following page shows the data structure used.
This list is implemented by indexing a large array of elements called ‘Nodes’. Node is actually declared as a structure in ‘vhdl.h’ header file which is not a standard ‘C’ header file but a developer developed file. Therefore each ‘Node’ contains information like name, type, location etc.

```
typedef struct {
        char        name[MAX_LABEL];    /* node name */
        int         type;               /* node type */
        int         ptr[MAX_PTR];       /*MAX_PTR =6*/
        rect        R;                  /* icon ref */
      }     a_node;

typedef struct {
        int         Xmin;
        int         Ymin;
        int         Xmax;
        int         Ymax;
      }  rect;
```

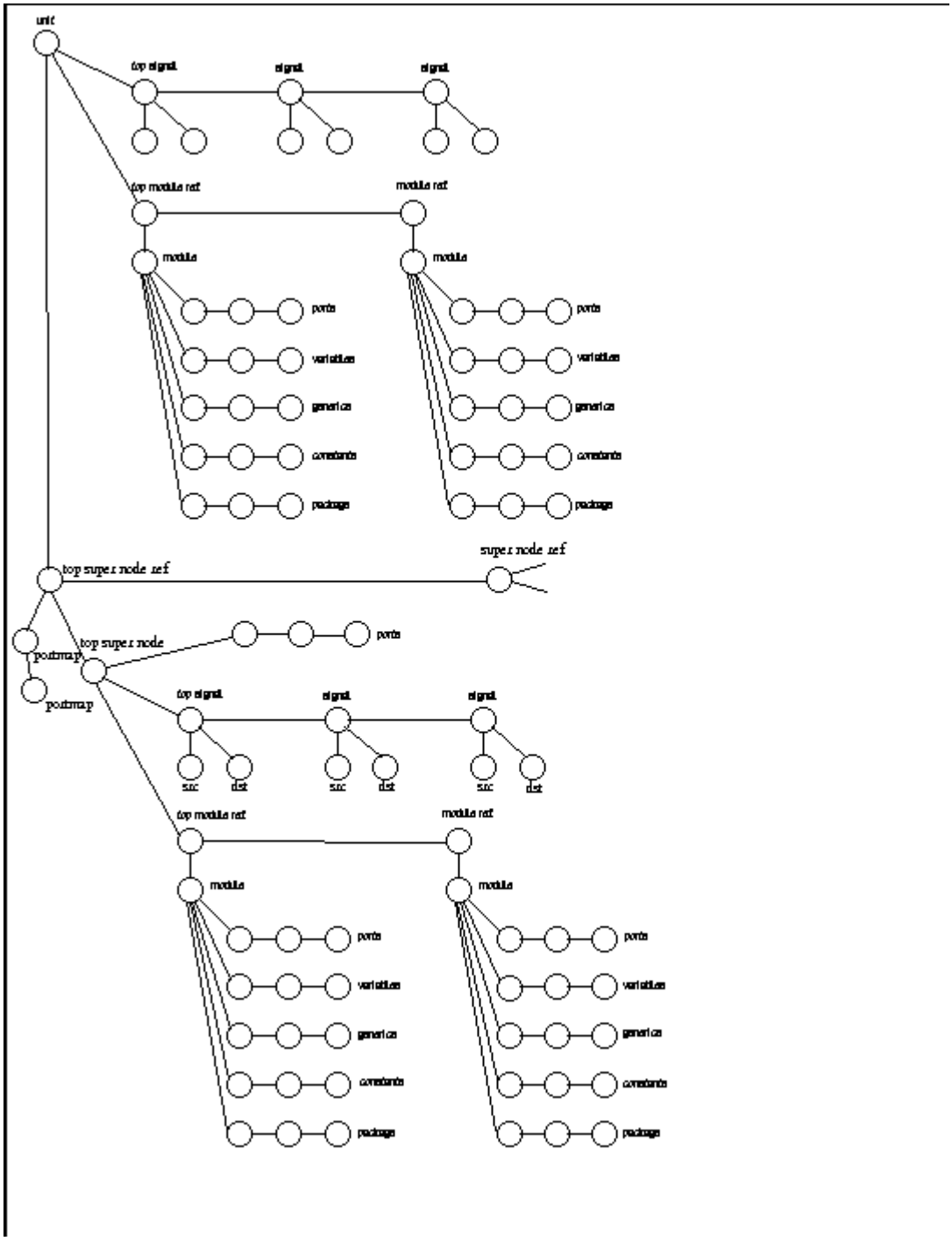

Figure 6.1 The Linked List Structure

Each ‘Node’ therefore contains six pointers storing various information. The linked list structure is shown in the following page. The only major change made to the already existing data structure is to move the ‘Library’ pointer which was Pointer 3 under Unit menu was removed and the new Library related pointer is added to the Module. Pointer 3 of the Module is used for Library now.

Other changes involve the ones related to the Base type, DimStart, DimEnd etc to accommodate to the fact that ‘Std_Logic’ is now available as one of the data types.

Refer to the “macros.h” to study this linked list structure in detail.

Parameter Name	Program Name	Description
Mvhd	mvhd	Creates the dummy entity for the process
VCADANALYZER	vhdlan	Synopsys Analyzer
VCADSIMULATE	vhdldbx	Synopsys Simulator
VCADSHELL	dc_shell	Synopsys Synthesizer
VCADTBENCH	/project/tpals3/VHDL/tbench /bin/tbench	Synapticad Software
VCADPRIM	/usr2/cad/modas/primlib	Directory where pro primitives are stored

Figure 6.2 Various Parameters

APPENDIX C

File Description

- **Program**: modas_Top.c

This is the top level file for the Modeler's Assistant VHDL Cad System. This is the file that creates the entire pop up menus and sets the dimensions and the resources of the windowing system. This is the main file where the application main loop starts.

- **Program**: units.c

This is the file that defines the callbacks different menu items. This file contains all the information about the functions the Modeler's Assistant uses. This file has been modified to allow the new features in the tool.

TO ADD NEW MENU ITEM:

1.Add the menu item in the appropriate place in menus.c. If the user is creating a new menu list then add it at the end of the already existing ones.

2.Then go to 'vhdl.h ' header file and add the new menu in the 'define menu' number list.

3.Then write the appropriate case statements required in units.c. Follow any menu in 'units.c' to make the necessary changes or additions to the tool.

- **Program**: primitives.c

This is the file contains functions relating to creation and instantiation of primitives.

- **Program**: variables.c

This file contains functions for manipulating variables and constants.

- **Program**: ports.c

  This is the file that stores the information about the PMG ports into the database of the PMG. This file contains functions relating to port naming, port printing, printing the port mode and type.

- **Program**: menus.c

  This is the file that stores the information about the menus that appear in Modeler's Assistant. This is the file that creates the entire menu list for functions based in the PMG. Edit the file if you need to add any menu/delete any menu .Simultaneously edit units.c Resource specification for creating menus is also dealt here

- **Program**: misc.c

  This is the file that has most of the functions for generating the VHDL code. This file has functions relating to X windows to set up the windowing structure. The entity and the architecture for the PMG are generated here. The file also has functions for calling the analyzer, simulator.

- **Program**: mvhd.c

  This is the file that creates the dummy entity for a process. This dummy entity is created using the "load_module" function and extracting the required information like portname, generic info from the linked list. It is compiled as follows:
  compiled as "cc my_pmg.c mvhd.c -g -o mvhd"

- **Program**: my_pmg.c

  This is the file that stores the information about the PMG into the database of the PMG. i.e. this is the file that creates the entire linked list of

nodes based on the PMG. The files needed are *.mod and *.unt of the PMG created by the Modeler's Assistant.

- **Program**: supernodes.c

  This is the file that stores functions about the creation of supernodes, supernode collasping and explosion i.e. file contains functions associated with creating, editing , and manipulating supernodes.

- **Program**: statement.c

  This is the file has functions about making the signal and other functions for manipulating the signal information.

- **Program**: expr.c

  This is the file has a function which helps determine the type of port/variable/generic/supernode/constant.

- **Program**: generics.c

  This is the file that contains functions to add/delete a generic,print generic,its type and has functions which perform printing of value of constant,signal etc.

- **Program**: bugs.c

  This is the file that stores the functions to add and delete various packages and has some initiation functions.

- **Program**: modules.c

  This is the file that stores the information about the modules and primitives of the PMG. It has functions relating to loading and saving primitives and modules.

- **Program**: vcad.c

  This is the file all the required functions for taking the keyboard events Carriage return key, a 'y'/'n' etc. It has the most important AppMainloop function which transfers control of the application to Xt. Xt drives the widgets in response to events and interacts with application only at times the application has arranged ahead of time.

Vita

Soumya Narnur was born in Hyderabad, India on January 1976.She graduated with a bachelor of Engineering Degree (B.TECH) in Electrical and Electronics Engineering from Jawaharlal Nehru Technological University, Hyderabad, India in June 1997. She attended Virginia Polytechnic Institute and State University for her Master of Science and received it in August, 1999. She has been employed with Intel Corporation, California since August 1999.