VHDL User's Guide

by
David C Blight
VLSI Research Laboratory
Department of Electrical & Computer Engineering
University of Manitoba
Winnipeg, Manitoba, Canada

Contents

1 Introduction

This is the user guide for version 1.2.0 of the vhdl tools.

This program will read a subset of the VHDL language and produce an edif netlist for use with Cadence, xnf netlist for use with the XILINX FPGAs, or optionally simple simulation of designs using waveform assignment statements.

This software is still in development stage. There plenty of bugs, and lots of unimplemented features. Due to the complexity of the VHDL language, not all the language is supported. Only those features that I have needed have been implemented. Others are slowly added with each release.

VHDL (Very High Speed Integrated Circuits Hardware Description Language) is textual means of describing systems design information. Structural descriptions may be converted into EDIF netlists which may used with the Cadence Edge tools to produce an IC layout. Behavioral VHDL descriptions may be simulated using the VHDL simulator.

This guide is an introduction to the use of the VHDL tools implemented at the University of Manitoba. This guide does not discuss the internal operation operation of the tools, but only operation from a user's perspective. The implementation aspects of these tools are discussed in [10] , [11].

The first section gives a brief overview of the VHDL language. The most commonly used aspects of the language are briefly reviewed. The following sections discusses the different VHDL tools incorporated into the VHDL **analyzer** program.

1.1 VHDL Language

The VHDL language is an IEEE standard language [**?**]. This section gives a brief introduction to the language. An example of a VHDL description is available in section 8. This section will consider only those aspects of the language which are currently supported, akthough not all supported features are discussed. This section does not describe the total VHDL language, but rather introduces only enough to get the first time user started. More explicit VHDL information may be found in [2].

The VHDL design description consists of a sequence of design units. Each design unit is of one of the following types :

- A Package contains a set of declarations and specifications which may be shared by other design units.
- An Entity (also called entity interface) describes the externally visible connections of a design. All information declared in an entity is visible to all architecural bodies associated with the entity. Ports are used to represent externally connections, and generics are used to represent design values.
- An Architecture (also called entity body) contains the design implementation information. For a structural description this will contain components and connectivity information. For a behavioral description, this will contain signal assignment statements.

Like programming languages, VHDL offers many of the similar features and constructs to aid in system design. Many predefined types are available to the user in the package standard which contains the basic logical and numeric types. VHDL allows the user to define abstract types in addition to those found in the predefined package "standard". Scalar types represent integer and floating values. Arrays types may be used to represent collections of similar typed data values. Record types are used to represents collections of non homogeneous data tpyes.

Structural descriptions are used to describe connectivity information of a design. Connectivity may be expressed by set of node declarations representing nets, and a set of component instantiation statements representing components, either hierarchal objects or standard cells. Component instantiations can be nested in for and if statement loops.

Behavioral descriptions are represented as a set of signal assignment statements. Each statement describes a relation between a node (called target) and an expression containing other nodes. Changes in expression values cause the target value to change after an appropriate delay.

1.2 VHDL Tools

The tools implemented for use with VHDL are all incorporated in the the single program **analyzer** and for the most part the operation is transparent to the user. It is important to be aware of the distinctions since different errors are reported from different programs. The different tools are :

1. The VHDL Analyzer is responsible for parsing the input VHDL description and converting the VHDL into an internal format (called VIN). VIN may be stored in libraries and shared with other designs.

2. The VHDL Reverse Analyzer may convert VIN into VHDL code functionally equivalent to the VHDL code which produced it.

3. The elaborator (also referred to as the structural compiler) is responsible for transforming the VIN format into a circuit representation. After elaboration, a model of the described circuit is available for netlisting or simulation.

4. The EDIF netlister produces an EDIF "netlist" view of the elaborated circuit representation. This produces the edif netlist which may be read into Cadence, or any other tools supporting EDIF.

5. The VHDL netlister produces a flat simple structural VHDL description of the circuit representation. This is different from the reverse analyzer which simply translate VIN to VHDL. This is usefull for interfacing to VHDL programs utilizing only a small subset of the VHDL language such as the Xilinx VHDL compiler.

6. The Xilinx netlister will produce an xnf (XILINX proprietary) netlist. This netlist may be used to program the xilinx family of FPGAs.

7. The VHDL Simulator simulates the VHDL behavioral description. A structural hierarchy and behavioral models for standard cells may be mixed to allow simulations of structural descriptions.

2 Program Operation

This section discusses the operation of the **analyzer** program which not only performs the functions of the VHDL analyzer, but coordinates the activities of the other VHDL tools.

The first subsection discusses the command line arguments the analyzer program. The most common sequence of actions are executed by a single option.

Not only may options be specified on the command line, but information may be stored in a .vhdl initialization file. The format of this is discussed in the next subsection. The final subsection lists the various files used and produced by the by the analyzer program.

2.1 Command Line Options

The **analyzer** program may be called with one and only one of the following options.

-I initialize symbol table, create prefined packages, and create syntax tree. This is used only during installation of the program, and is noramlly disabled.

-G file_name analyze filename, elaborate, EDIF and VHDL netlist. The file_name must have a .hdl suffix.

-S file_name analyze filename, elaborate, and simulate the specified file_name.The file_name must have a .hdl suffix.

-M file_name read the library file, elaborate, EDIF and VHDL netlist. The file_name must have a .vin suffix.

-X switch netlist format to xilinx. Must be used with -G option.

-P silos netlist. NOT supported this release.

2.2 Initialization File

The analyzer program after reading the command line options, read a .vhdl file in the current working directory. This is file exists, certain default values may be overwritten. The file contains a sequence of commands, one per line. The following cammands are valid.

VHDL_START This specifies the start of the .vhdl file. If this is not the first command, the file contents are ignored.

SET *library file_name* This command is used to set the specified directory to the proper directory. Reference, Work, and Simulation are three predefined libraries.

CLEAR *library* remove library entry.

SIMULATION NOISEY print all simulator assignments. Useful in debugging.

SIMULATION QUIET Default simulator assignment option.

CORE DUMP enable core dumps.

NO LOG disbale printing of log information.

BANANARAMA set special flag. Normally disabled.

VHDL_END Specifies the end of the initialization file. All commands after this are ignored.

2.3 Files

The following files are used or produced by the analyzer program. Prefix xxx refers to abritrary file names.

.vhdl user defined initialization file.

xxx.hdl VHDL soruce file.

xxx.vin VIN library file.

xxx.vhdl VHDL netlist.

xxx.edif EDIF netlist.

xxx.sim Simulation output.

vhdl.log Log file for analyzer program.

core core file sometimes produced.

3 Using Cadence With VHDL

This section describes using the structural compiler and EDIF netlister for use with the Cadence Place and Route tools. This section does not discuss the operation of the Cadence tools. User's should be familar with these tools before reading any further.

3.1 Component Declarations

The package components in the default reference library contains most of the component declarations for cells in the University of Manitoba standard cell library (CMOS3DLM). Each component declaration has an associated attribute "CDS_INFO" which contains the following information :

num_ports defines the number ports on the cell. This includes those not visible in VHDL such as power and ground.

cell_name define the cadence blockname of the standard cell. This is case sensitive.

In addition, for each port, the following information is defined

vhdl_name identifies the VHDL port name.

cds_name specifies the cadence port name.

port_type specifies the cadence purpose for the port. Values are one of POWER, GROUND, CDS_SIGNAL,or DUPLICATE. Duplicate implies the port is logically the same as another port on the cell.

port_dir specifies the port direction. value may be one of CDS_INPUT, CDS_OUTPUT, or CDS_INOUT.

Likewise the attribute "CDS_SIGNAL_INFO" is defined for signals. Two signals, VDD and GND are defined in the package components. The fields of attribute "CDS_SIGNAL_INFO" are :

global specifies whether the signal is global or not.

signal_type specifies the type of signal. Allowed values include : CDS_SIGNAL, CDS_GROUND, or CDS_POWER.

cds_name specifies the name given to the signal. This is used to use signal names not allowed in VHDL such as "vdd!".

3.2 EDIF Netlist

The EDIF netlist generated is a flat netlist corresponding to the top level of hierarchy in the design. The netlist uses the netlist constructs of the EDIF 2 0 0 standard and supports the properties of Cadence's edifin program. A Cadence autoLayout represenation may be created from the edif file by running the edif200_to_cds program. If not errors are found, an autolayout representation is created. This may be used in the Place and Route environment in a similar manner to any autoLayout generated from a schematic.

The operation of the Place and Tools is not discussed here.

3.3 Silos Extraction

After generation of the final layout, it is wise to perform a simulation on the design. The prlabel skill program should be run on the layout, to create pins with the net names on the appropiate wires. This will allow net names to be visible in the simulation file. Extract the silos simulation deck using the layout_silos_rules_2.1 and netlist the extracted rep. The simulation should not be performed from within Cadence since not schematic exists to select waveforms from.

4 VHDL Simulation

This section discusses the operation of the VHDL simulator. Emphasis is placed on the simulation of structural descriptions using behavioral standard cell entities.

4.1 Component Declarations

In order to simulate a structural design hierarchy, behavioral models must be used with the standard cell components. This are already define and available in the predefined simulation librayr which is automatically loaded when the analyzer is run with the -s command line option. The library may be replaced with another library by setting the SIMULATION library in the .vhdl file.

4.2 Simulation Input

No external means of specifing test vectors for the circuit currently exists. All values must be applied using signal assignment statements.

4.3 Simulation Output

Simulator output is written to the .sim file corresponding to the current design. Only values are written by default that correspond to nodes in the top level of hierarchy. Optionally is the attribute SIM_MONITOR exists for a node, it's value is also displayed.

5 Error Guide

this section discuuses the error codes generated by the VHDL compiler. Errors number greater than 500 reflect errors could after compilation. All error codes over 1000 reflect internal errors of the compiler and should be reported.

0. A reserved word was expected but not found.
1. Syntax error occured. Check syntax at error point.
2. The subtype declaration contained a type reference was not able to be resolved.
3. The subtype declaration contained a type reference was not able to be resolved.
4. The expression in a constant declaration did not match the specified type.
5. The range of scalar type declaration was not a scalar of proper type. Check expression in range.
6. The expression in a signal declaration did not match the specified type.
7. A delimiter was expected but not found. If this error is found in a component instantiation statement, check whether the component name corresponds to a component declaration.
8. An identifer was expected but not found.
9. Aggregates in targets of conditional signal assignment statements are not supported.
10. A subtype indication contained an array range for a non array type (eg BIT(0 to 2)). Make sure type is an array.
11. An operaor was expected by not found.
12. The name (signal,constant,variable) was not correctly specified. Possible undeclared identifier.

13. Attributes not supported in range. This is an allowed syntax in VHDL 1076, but not yet supported.

14. Positional aggrgate entity not allowd after named aggregated entity.

15. invalid index in array constraint.

16. the specified attribute was not declared. This error is detected during an attribute specification.

17 A operator was used with incompatible operands.

19 An invalid or missing declaration or specification was found.

20 The enumeration Literal is not part of the specified type.

21 Lexical Analyzer error. The token produced does not match VHDL define types.

22 A name was used in a use statement which did not correspond to any design entity.

23 The specified name is not a constant signal or variable.

24 The index of an invalid name was requested. Check name syntax.

25 Wrong number of operands in a component instantiation statement. Check component declaration statement and instantiation statement to make sure they match.

26 Component in a component instantiation was not declared.

400 Invalid function in evaluate operation. A function was used that is not implemented.

500. Incompatible sizes were found in attribute initialization. Probably caused by incorrect type in attribute specification. This error should have been detected earlier.

501. A attribute was not initialized with a value.

502 Invalid use statement, probably caused by error 22.

503 An attempt to connect nodes together with different sizes was made. Probably in component instantiation statement.

504 The index of a node or variable was outside the bounds of the variable. Check index value.

505 The index of a node or variable was outside the bounds of the variable during the creation of a connection. See error 805, 504.

701. An attempt was made to xilinx netlist a description which does not have a architecture as top level in hierarchy.

800. net was not a global net. When the port of a component which is a power or ground is added to the edif output, the global net with the same name was not located.

801 No CDS_INFO attribute was found for the specified component.

802 The specified component name was not found. This is typically the result of a 801 error.

803 unable to connect to node. The edif netlister was not able to resolve a component port.

804 port was not in component declaration. A port was named in the CDS_INFO attribute for a component, but it was not found in the component declaration.

805 unable to make gloabl connections for component.

900. No name for the edif netlist cell was specified, default name of NEW_CELL was used.

1000. Internal parsing error (psyntax end).

1001. Internal parsing error (epsilon rule).

9999. Internal error in program. Could be anything, Do not waste your time trying to debug this one, this is as bad as it gets.

6 Program Bugs

This section contains a list of currently known bugs for the VHDL tools. Bug numbers are kept contant across different versions of the programs.

5 Bug in lexical analysis. Certain lexical elements ending at end of buffer cause incorrect results. Syntax error usually appears as a result, which is incorrect. Lexical analysis errors are being included in analyzer error log which should help eliviate this. Usual action is to add spaces to beginning of input file.

7 Memory allocation error. Not all simulation memory is properly discarded when finished. Long simulations may run out of memory.

8 Not all user errors cause message to be included in error log. Please report core dumps and error 9999.

13 No type checking is performed on component declarations, and function calls. This may lead to unexplained errors later on.

22 Some incorrect syntaxes in input deck give rather cryptic messages. More help should be provided. Example component instantiation statement.

23 A missing entity_statement_part after a BEGIN reserved word will cause an internal error, and core dump.

7 Unimplemented Features

1. Bus resolution function not supported.
2. Operator overloading not currently supported
3. Configurations not supported.
4. Blocks not supported (soon to be implemented)
5. Access types supported but no where to use them yet
6. File Type support but no i/o functions yet.

7. Package Bodies implemented but not needed yet.

8 Design Example

This sections gives an example of VHDL code describing a simple n -bit ALU. The package components is not shown.

```
--
--       test vhdl program for analyzer 1.00
--


--
-- Define a package misc which contains
-- a type declaration used by otherdesign units
--
package misc is

   type control_struct is record
      con0   : bit;
      con1   : bit;
   end record;

end misc;

--
-- The use statement tells the analyzer to
-- use the packages components, and misc.
-- components is assumed to be in an accessable
-- library.
--
use components,misc;

-- Begin Entity declaration.
--     N reprsents the number of bits in alu
--     Ports represent external connections of ALU
--
entity alu_example is
   generic
      (
```

```
      N : natural := 3
      );
   port
      (
      A : bit_vector(0 To N);
      B : bit_vector(0 To N);
      Output : bit_vector(0 to N);
      Con : control_struct;
      Cin : bit;
      Cout : bit
      );
end alu_example;

--
-- The architecture part contains the connectivity
-- information for the alu
--
use components,misc;
architecture aalu_example of alu_example is
   signal B_not    : bit_vector(0 To N);  -- signal declarations
   signal X1,X2,X3 : bit_vector(0 To N);
   signal Carry    : bit_vector(0 To N-1);

begin
   for i in 0 To N generate
      inv port map(B(i),B_not(i));        -- component instantiations
      nand2 port map(B(i),Con.con0,X1(i));
      nand2 port map(B_not(i),Con.con1,X2(i));
      nand2 port map(X1(i),X2(i),X3(i));

      if i = 0 generate
         fadd port map(a(0),X3(0),Cin,Output(0),Carry(0));
      end generate;

      if (i > 0) and (i < N) generate
         fadd port map(a(i),X3(i),Carry(i-1),Output(i),Carry(i));
      end generate;
```

```
      if i = N generate
         fadd port map(a(N),X3(N),Carry(N-1),Output(N),Cout);
      end generate;

   end generate;
end aalu_example;

-- chip1 is the description of the chip
-- it incorporates aalu_example through
-- a component declaration statement and
-- component instantiation statement
--
--  Inpads and out pads are also specified
--
use components,misc;
entity chip1 is
   generic
      (
      N : natural := 3
      );
   port
      (
      A : bit_vector(0 To N);
      B : bit_vector(0 To N);
      Output : bit_vector(0 to N);
      Con : control_struct;
      Cin : bit;
      Cout : bit
      );
end chip1;

use components,misc;
architecture version1 of chip1 is
   subtype shit_for_brains is bit_vector(0 to n);
   signal cout_tmp : Bit;
   component aalu_example
```

```
    generic
       (
       N : natural
       )
    port
       (
       A : bit_vector(0 to n);
       B : shit_for_brains;
       Output : shit_for_brains;
       Con : control_struct;
       Cin : bit;
       Cout : bit
       )
  end component;
  signal xyz,xyz1 : bit;
begin

  xyz <= xyz1 after 10 ns;           -- signal assignment statement
  ALU  : aalu_example generic map(N)
     port map(A,B,Output,Con,Cin,cout_tmp);
  CON0 : inpad port map(Con.con0);
  CON1 : inpad port map(Con.con1);
  CIN  : inpad port map(Cin);
  COUT : outpad port map(Cout);
  for i in 0 To N generate
     inpad port map(A(i));
     inpad port map(B(i));
     outpad port map(Output(i));
  end generate;
  DUMMY : inv port map(cout_tmp,Cout);
  vddpad;
  gndpad;
end version1;
```

References

[1] IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-1987.

[2] Armstrong J. A., “Chip-Level Modeling with VHDL,” Prentice Hall, 1989.

[3] Shahdad M., Lipsett R., Marschner E., Sheehan K., Cohen H., “VHSIC Hardware Description Language,” *IEEE Computer,* pp. 94 - 103, 1985.

[4] Silos II Logic and Fault Simulator, User’s Manual, Simucad Inc, 1988.

[5] Cadence EDGE system, CADENCE, 1989.

[6] HSPICE Users’ Manual H8801, Meta-Software Inc, 1988.

[7] EDIF Specification, Electronic Design Interchange Format, Version 1 1 0, EDIF Steering Committee, 1985.

[8] Kostiuk, Andrew R., *QUISC : An Interactive Silicon Compiler*, MSc. Thesis, Queens University, 1987.

[9] Liu, Bing, *BINGS THESIS TITLE*, MSc. Thesis, University of Manitoba, 1990.

[10] *VHDL Design Tools for IC Design* MSc Thesis, University of Manitoba, 1990.

[11] *Integrated Circuit Design using VHDL*, Canadian Conference of Electrical and Computer Engineering,1990.