Using SAVANT for VHDL Simulation

(Documentation for version 1.0)

Radharamanan Radhakrishnan and *Philip A. Wilsey*
Computer Architecture Design Laboratory
Dept of ECECS, PO Box 210030
Cincinnati, OH 45221–0030
savant@ececs.uc.edu

Copyright © 1995–1999 The University of Cincinnati. All rights reserved.

Published by the University of Cincinnati
Dept of ECECS, PO Box 210030
Cincinnati, OH 45221–0030 USA
Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions.

Chapter 1

Introduction

So you have successfully installed SAVANT, TyVIS and WARPED software distributions and are now ready to use the software for VHDL simulation. If you have not done this, please follow the installation directions in the README, INSTALL, and INSTALL-FOR-SIMULATION text files available in `savant` directory[1]. Please follow these directions and install the software before proceeding further.

This document attempts to give the reader step-by-step guidelines on how to perform (sequential or parallel) simulation of VHDL examples. The easiest method to do this is to use a VHDL example and walk the user through the steps. Atleast some VHDL knowledge is required to understand the following description. So without further ado, lets study the sample VHDL example (`test.vhdl`) that we will use to describe the simulation process:

1.1 Example Program - `test.vhdl`

```
-- ******************************************************************
-- Sample VHDL Simulation Demonstration Program - test.vhdl
-- ******************************************************************

1 library ieee;
2 use ieee.std_logic_1164.all;

3 library std;
4 use std.textio.all;

5 entity myEntity is
6  -- empty entity
7 end myEntity;

8 architecture behavioral of myEntity is
```

[1] Also available at `http://www.ececs.uc.edu/~paw/savant`

```
9   begin

10    myProcess: process is

11     variable inputOne : std_logic_vector(0 to 1) := ('0','0');
12     variable inputTwo : std_logic_vector(0 to 1) := ('1','1');

13     variable traceLine : LINE;
14     variable space : string(1 to 2) := "  ";

15     file RESULT_FILE: text open WRITE_MODE is "results.out";

16     begin

17       write(traceLine, To_bitvector((inputOne and inputTwo)));
18       write(traceLine, space);
19       write(traceLine, To_bitvector((inputOne or inputTwo)));
20       write(traceLine, space);
21       write(traceLine, To_bitvector((inputOne xor inputTwo)));
22       writeLine(RESULT_FILE, traceLine);

23       wait;

24     end process myProcess;

25 end architecture behavioral;
```

As can be seen from the VHDL source code, the example instantiates a process which performs `and`, `or` and `xor` operations (Lines 17, 19, and 21) on two standard logic vectors (`inputOne` and `inputTwo`). In addition, it writes out the result of the operations (after converting the standard logic vectors to bit vectors by calling the To_bitvector function that is part of the `std_logic_1164` library) to a file (`results.out`).

1.2 Guidelines for simulating VHDL Programs

The installation process will build `scram` (the SAVANT VHDL analyzer) in a user specified directory (usually in `savant/bin`). The analyzer has the following command line parameters that the user needs to be familiar with:

```
mymachine.home.edu <user_specified_dir>/savant/bin> scram
savant 1.0 (March 8, 1999)
Copyright(C) 1993-1999 The University of Cincinnati.
All rights reserved.
```

```
Usage <user_specified_dir>/savant/bin/scram -options <files>
Where options is one of the following:
  -debug-symbol-table    print out debugging info relating to symbol table
  -design-library-name   design library name
  -publish-vhdl          publish VHDL
  -publish-cc            publish c++
  -no-file-output        send publish_cc output to stdout instead of files
  -warranty-info         print information about (lack of) warranty
  -static-elaborate      statically elaborate the specified design
  -partitions            specify # of partitions for elaborated design
  -entity                specify entity to use
  -arch                  specify architecture to use
  -config                specify configuration to use (if needed)
  -alg                   specify static partitioning algorithm to use
  -v                     verbose output
  -help                  print this message
```

Assuming that the installation process completed successfully, the following steps detail the process by which an user can use the SAVANT software for simulating VHDL examples.

1. Alias `<user_specified_dir>/savant/bin/scram` to `scram`. This is just for convenience so that the user can access the `scram` executable from any directory.

2. The example VHDL file (`test.vhdl`) imports two libraries (Line 1 and 3: `ieee` and `std`) and uses these libraries just like include files in C/C++. However, the libraries need to be handled differently from normal include files. The `std` library gets automatically included by the `scram` parser and the user need not do anything about this library. However, the `ieee` library needs to be analyzed/parsed before the example VHDL file can be analyzed. To do this, perform the following steps:

Step 1: Locate the `std_logic_1164.vhdl` source VHDL file. This file needs to analyzed before `test.vhdl` can be analyzed. This file isn't distributed with savant as it is copyighted by IEEE and not freely redistributable. Note too, that versions of this file (and other IEEE libraries) shipped with commercial analyzers may vary in subtle or not so subtle ways from IEEE's definition standard library. For instance, the std_logic_1164 from Synopsys requires other libraries that define things that aren't part of the standard.

Step 2: Look in the `std_logic_1164.vhdl` source VHDL file to see if it in turn includes any other libraries. If it does, then recursively execute this step.

If you look at the `std_logic_1164.vhdl` source, you will see that it includes a package (`attributes`) from another library called `synopsys`. So we will need to locate the `attributes.vhdl` file which contains the "attributes" package used by the `std_logic_1164.vhdl` library.

Step 3: Having resolved the include dependencies of the libraries to the `attributes.vhdl` file, we are now ready to create the `synopsys` and `ieee` design[2] libraries.

Analyze the `attributes.vhdl` file into the `synopsys` design library by executing the following command:

```
scram -publish-cc -design-library-name synopsys attributes.vhdl
```

This will result in the creation of a design library named `synopsys._savant_lib` and the code generation of TyVIS compliant C++ source files in the current working directory. If you look inside this directory, it will contain code-generated VHDL files, C++ files and a Makefile. This directory is a self-contained unit and the user need not do anything further with this directory.

Step 4: Repeat Step 3 for the `ieee` design library. Since the `synopsys` library is now available, the `std_logic_1164.vhdl` source file can now be analyzed into the `ieee` design library. Hence, executing the following command:

```
scram -publish-cc -design-library-name ieee std_logic_1164.vhdl
```

will result in the creation of a design library named `ieee` and the code generation of TyVIS compliant C++ source files in the newly created design library directory. If you look inside this directory, it will contain code-generated VHDL files, C++ files and a Makefile. Again, this directory is a self-contained unit and the user need not do anything further with this directory.

3. We are now ready to analyze the example VHDL program, `test.vhdl` given that both `ieee` and the `synopsys` design libraries are now available. Execute the following command:

```
scram -publish-cc test.vhdl
```

4. This will result in the creation of the `work._savant_lib` directory. This is a work directory automatically created by `scram` and contains code generated VHDL and C++ files. It also contains the top level Makefile that is cognizant of the other two design libraries (`ieee` and `synopsys`). Now all the user needs to do, is to execute the following command:

```
cd work._savant_lib;
make depend; make
```

5. The execution of the aforementioned commands will result in the compilation of all the code-generated C++ files (including those in the `ieee` and `synopsys` design libraries)

[2]For more information about design libraries and how they are handled in SAVANT, please refer to the documentation on the library manager (available at `http://www.ececs.uc.edu/~paw/savant`)

and the creation of the executable (`work_Dmyentity_work_Dmyentity_Dbehavioral`). This will be the final simulation executable that the user needs to execute to run the simulation. Executing the following command:

```
work_Dmyentity_work_Dmyentity_Dbehavioral
```

will result in the initiation of a sequential simulation (assuming the SAVANT software was configured for sequential execution; the SAVANT software can be configured for parallel execution also[3]) of the VHDL source file. Since the original VHDL file writes the output of the logical operations to a file (`results.out`), the last step is to ascertain if the simulation executed correctly and the correct results were written to the file.

6. Looking at the `results.out` file, we see the following result:

```
00 11 11
```

So, is this result right? (`00 and 11`) results in `00`. (`00 or 11`) results in `11`. (`00 xor 11`) results in `11`. So, the output is correct implying that the simulation executed correctly and that concludes our little simulation demonstration.

Acknowledgments

This research has been conducted with the participation of many investigators. While not an complete list, the following individuals have made notable direct and/or indirect contributions to this effort (in alphabetical order):

Jeff Carter, Harold Carter, Praveen Chawla, Malolan Chetlur, John Hines, Herb Hirsch, Dale E. Martin, Timothy J. McBrayer, Greg Peterson, Phani Putrevu, Radharamanan Radhakrishnan, Umesh Kumar V. Rajasekaran, Dhananjai Madhava Rao, Al Scarpelli, Mike Shellhause, Krishnan Subramani, Swaminathan Subramanian, Narayanan V. Thondugulam, and John Willis.

We wish to express our sincerest gratitude for the time that you spent reviewing and commenting on our designs. This research was supported in part by the Wright Laboratory at Wright-Patterson AFB, DARPA, and MTL Systems, Inc. Thank you for your support.

[3]Please refer to the INSTALL-FOR-SIMULATION document for instructions on how to configure the SAVANT software for parallel execution.