VHDL Synthesis System (VSS)
User's Manual
Version 5.0

Loganath Ramachandran
Viraphol Chaiyakul
Daniel D. Gajski

Technical Report #92-52
June 1, 1992

Dept. of Information and Computer Science
University of California, Irvine
Irvine, CA 92717
(714) 856-8059

ramachan@ics.uci.edu

Abstract

This report provides instructions for installing and using the VHDL Synthesis System (Version 5.0). VSS is a high level synthesis sytem that synthesizes structures from an abstract description, written with VHDL behavioral constructs. The system uses components from a generic component library (GENUS). The output of VSS is in structural VHDL and could be verified using a commercial VHDL simulator. The designer can control the synthesis process by providing different resource constraints to the system. VSS is also capable of producing different architectures which can be selected by the designer.

Contents

List of Figures

1 Overview

VHDL Synthesis System (VSS) is a high level synthesis sytem that produces a structure netlist of microarchitectural components from an abstract behavioral description. VSS accepts input descriptions written with VHDL behavioral constructs. The system uses components from a generic component library during the synthesis process. The output of VSS is in structural VHDL and could be verified using a commercial VHDL simulator. The designer can control the synthesis process by providing different resource constraints to the system. VSS is also capable of producing different architectures and the designer can select one of the architectures for synthesis.

VHDL is an acronym for VHSIC(Very High Speed Integrated Circuit) Hardware Description Language. The VHDL standards are defined in document[8]. The VHDL constructs not currently supported by VSS are listed in appendix A of this report.

This report provides instructions for installing and using the software in your site and is organized as follows: the next section provides detailed instructions for installing the software. Section 3, gives a brief description of the overall system. In Section 4, we show the different commands that can be invoked. Section 5 and 6, give details of the component library and creation of the resource constraint file respectively. We finally show a simple walkthrough tutorial to make it easy to understand and use VSS.

2 Installation Instructions

This version of VSS has been tested on SUN-4 platforms with SUN OS 4.1.1. The VSS software requires around 10 Megabytes of disk space. The interactive displays, require XWindows (Release X11R5) and Motif (Release 1.1). The control synthesis requires some of the programs from Octtools (distributed by UC Berkeley).

In order to load the software on your system complete the following steps:

1. Insert the VSS Distribution Tape into the tape drive of a SUN-4 machine.

2. Move to your home directory
   % cd $HOME

3. Create a directory for installing the VSS software. We shall refer to this directory as *release_dir*. Make sure that *release_dir* has the appropriate write permissions. Also change the working directory.
   % mkdir release_dir; cd release_dir

4. The VSS directory will be loaded into a subdirectory named *vss_ver5*. (Ensure that no other files with the same name exist). The VSS release can now be loaded by using the command
   % tar xvf /dev/rst8

5. Do not forget to remove your tape from the drive. You can check to see if the installation was successful by listing the files that were loaded and comparing it with the files shown in the directory hierarchy (Figure 1).

6. We now have to setup some of the script files. Edit the file *vss_ver5/scripts/vss_script*. This file defines a few environmental variables. The original file (on the release tape) will contain the following:

```
# Fill the name of the directory where VSS is installed.
setenv RELEASE_DIR    ....


# Fill the name of the directory where octtools is installed
setenv OCTTOOLS_DIR   ...

setenv OCTTOOLS      $OCTTOOLS_DIR
setenv CONTROL_SYN $RELEASE_DIR/vss_ver5
setenv GENUS_VHDL_DIR $RELEASE_DIR/vss_ver5/vhdl_models
setenv GENUS_DTAS_DIR $RELEASE_DIR/vss_ver5/dtas_model
set path = ($path $RELEASE_DIR/vss_ver5/bin)
setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:${RELEASE_DIR}/vss_ver5/libraries
xrdb -merge  $RELEASE_DIR/vss_ver5/scripts/Xdefaults
```

Edit this file to reflect the proper pathnames. Fill in proper pathnames of the VSS release directory and the octtools directory. After the above changes the first four lines of the script file would look like:

```
# Fill the name of the directory where VSS is installed.
setenv RELEASE_DIR    /cz/ua/logi/release_dir
# Fill the name of the directory where octtools is installed
setenv OCTTOOLS_DIR   /cz/ua/octtools
```

7. This file *vss_ver5/scripts/vss_script* must be sourced into the shell before invoking VSS.

   % source vss_ver5/scripts/vss_script

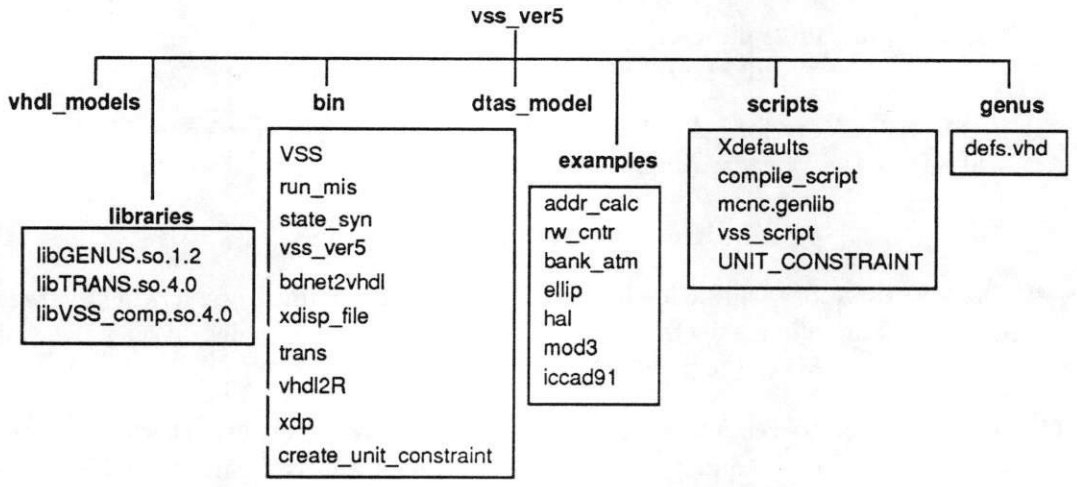


Figure 1: VSS Release - Directory Hierarchy

3 VHDL Synthesis System

The VSS system is a synthesis framework that contains a collection of tools. The overall picture of the VSS system is shown in Figure 2. The system can be divided into four set of tools based on their functionality. These sets are: (i) VHDL preprocessors, (ii) core synthesis tools, (iii) display tools and (iv) User Interface.

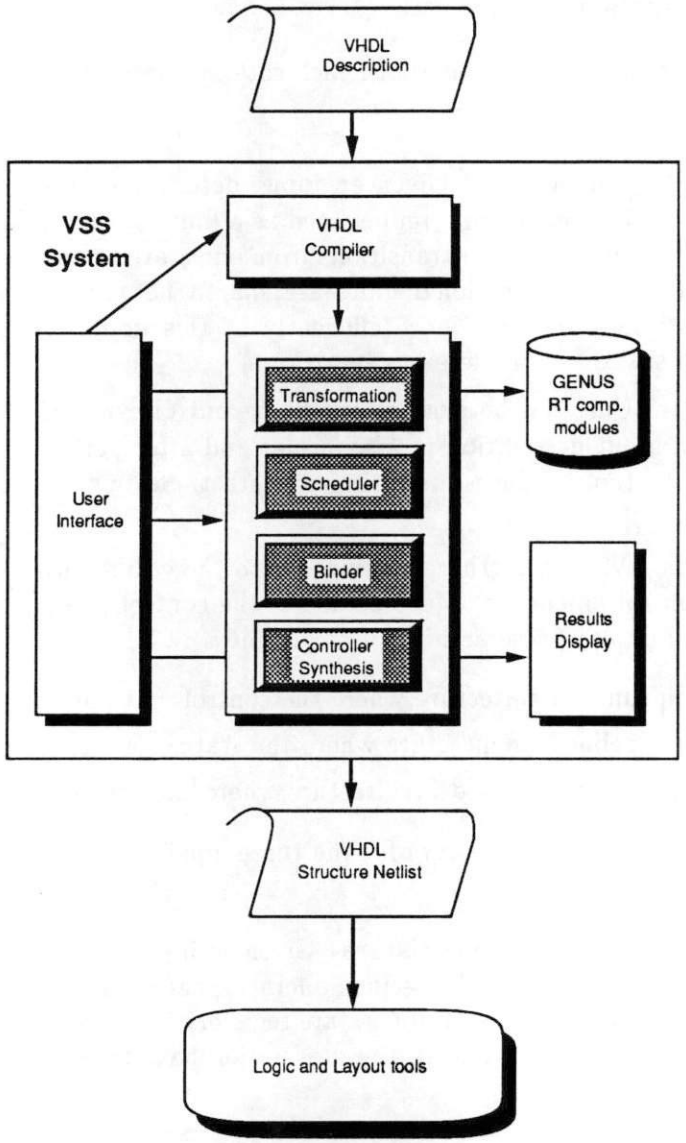


Figure 2: The VHDL Synthesis System

3.1 VHDL preprocessor

Currently, the VHDL preprocessor serves two important functions. These functions are:

- Handle multi-process and multi-block descriptions - The input description could contain mutliple processes or blocks. Each process gets synthesized into a CU-DP pair. In order to simplify this

synthesis task the description must be divided into multiple entities with each entity containing a single process or block. This division of a multi-process or block description is done by the VHDL preprocessor

- Structured Modeling Guidelines - The VHDL description must adhere to certain modeling guidelines described in [1, 2]. The preprocessor checks for adherance to these guidelines.

3.2 Core Synthesis Tools

VSS contains two major synthesis tools, each catering to a different design style. These design styles are:

- **FSM and a Datapath** - Given an input description this tool produces a multi-state design consisting of a Control Unit (represented as a finite state machine) with a datapath. The design consists of many states and transitions from one state to another performing different operations in each state. The functional units are shared between different states. Thus the datapath components may be used more efficiently in this design style, but the design takes a larger number of states to complete one iteration.

  The Control Unit can be synthesized by a control synthesis tool called **state_syn**. State_syn accepts an input description in *kiss* format and after performing state encoding and logic minimization the tool produces an optimized netlist. State_syn uses some of the programs available in Octtools [3].

  This design style can further be divided into three different architectures, each with a different level of control pipelining. More details of the control pipelined architectures produced by VSS is available in [4]. These architectures include:

  - non_pipelined architecture where the control unit and the datapath are not pipelined
  - status_pipelined architecture where the status signals are pipelined
  - control_status_pipelined architecture where the control and status signals are pipelined.

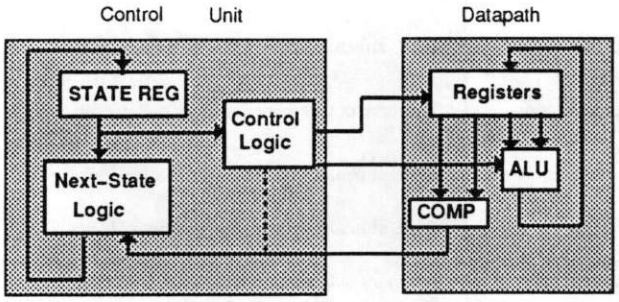
  Figure 3 shows the architecture for the three pipelinining methods. The designer can select any one of these architectures.

- **Designs with zero state registers** - Given an input description, VSS can produce a maximally parallel design. Therefore all specified operations are completed in one clock cycle or state. Hence, the control unit does not contain a state register. However, this design style may contain a large datapath since all operations in the description have to be performed in parallel.

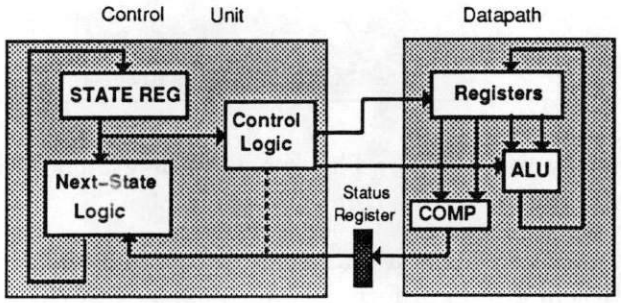
3.3 Display Tools

After the synthesis process completes, different types of results and statistics are provided by the core synthesis tools. The table in Figure 4 shows the type and format of results produced by the synthesis tools. These results could either be in textual or graphical form. In order to view the results two different display tools are released along with VSS.

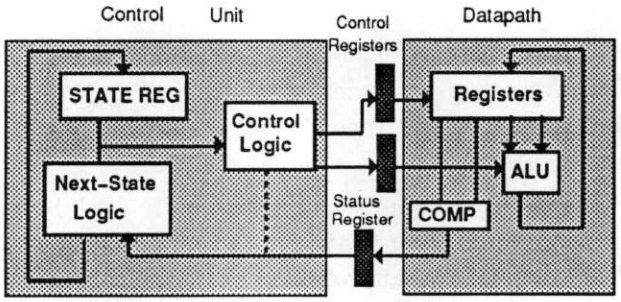
- Textual Display tool (**xdisp_file**) - Used for displaying all textual results. (A simple text editor could be used as a substitute for this tool).



(a) Non Pipelined Architecture



(b) Status Pipelined Architecture



(c) Control Status Pipelined Architecture

Figure 3: Control Pipelining Schemes

Result filename	Description	Format	View with
V_P1_final.dgm	CDFG of description	dgm	xdp
V_P1.state_info	Description of Actions in each state	BIF like	xdisp_file
V_P1_cu.kiss	KISS format description of control unit	KISS	xdisp_file
V_P1_cu_process.vhdl	VHDL process description of control unit	VHDL(proc)	xdisp_file
V_P1_dp_struct.vhdl	VHDL structural Description of Datapath	VHDL(str)	xdisp_file
V_P1_dp_summary	Summary of Components used in Datapath	text	xdisp_file
V_P1_struct.vhdl	VHDL structural description of the design	VHDL(str)	xdisp_file
V_P1_cu.vhdl	VHDL structural description of Control Unit	VHDL(str)	xdisp_file
V_P1_cu_summary	Summary of Components used in ControlUnit	text	xdisp_file

(1) Assuming V_P1.vhdl is the input filename

*(2) All the above files except *.dgm files will be found in a separate subdirectory called 'RESULTS'*

Figure 4: VSS Results

- Flowgraph Display tool **(xdp)** - This is one of the graphical display tools and is used to display the flowgraph produced by the core synthesis tools.
- Netlist Display tool - This is a graphical display tool which can be display the CU-DP VHDL structural netlist in graphical form. (Not available with current release).

3.4 User Interface

The User Interface (UI) provides users with a simple and elegant mechanism to control the synthesis process. The UI displays a top level view of the design to the user. This top level view shows the individual blocks and processes and their interconnections with global signals. A picture of the User Interface is shown in Figure 5.

Using this top level view, designers can select individual blocks or processes to be synthesized. The various options available during synthesis can be specified interactively. The results of synthesis can also be viewed by clicking the appropriate buttons on the User Interface. The control of the UI mechanism on the various tools is shown in Figure 2.

4 VSS Commands

The VSS system can be used in two different modes. (i) **the shell mode**, where the commands can be issued directly from the UNIX shell. This mode can be used for simple non hierarchical behavioral descriptions consisting of a single process or block. (ii) **the interactive mode**, which provides a flexible user-interface for invoking various VSS commands.

Before invoking VSS, the vss_script must be sourced into the shell.

% source <release_dir>/vss_ver5/scripts/vss_script

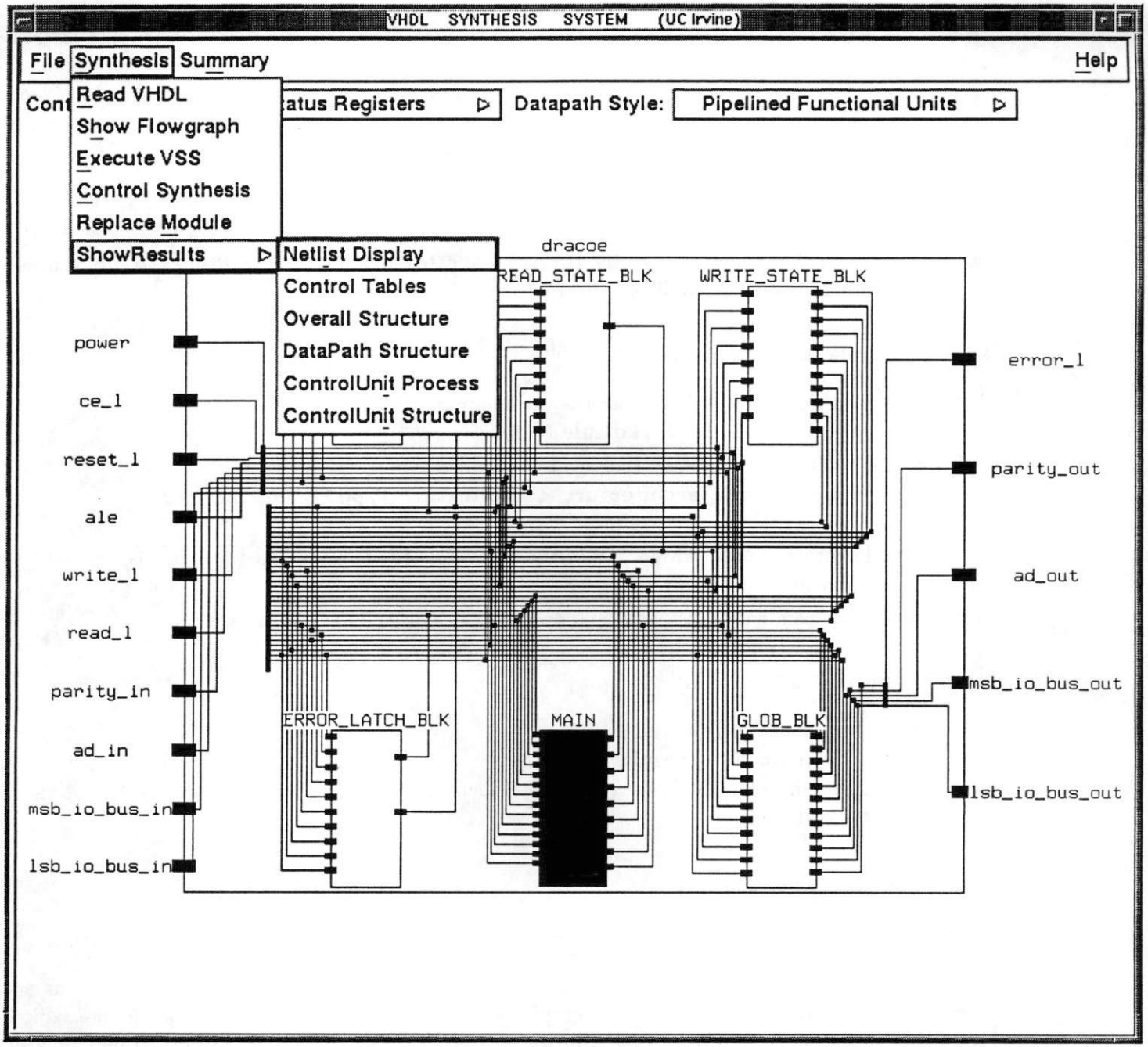


Figure 5: The VSS User Interface

The <release_dir> refers to the pathname where VSS was installed. (see section 2).

Before executing the following shell commands, ensure that you change the working directory, to the location where the VHDL file was created. If a VHDL behavioral description has been created in a file $HOME/models/a.vhdl, the following command needs to be executed

% cd $HOME/models

A *UNIT_CONSTRAINT* file that provides resource constraints is required in the same directory as the model. This file specifies the maximum number of functional units of a particular type that can be used during the synthesis process. A program called **create_unit_constraint** has been included in the release. This will provide a way to interactively define the components. After adding the required components to the database the data must be saved on a file called *UNIT_CONSTRAINT*.

4.1 Shell Mode

The commands in the shell mode are divided into two categories. (i) commands for the core synthesis tools. (ii) commands for results display.

4.1.1 Commands for Synthesis

To invoke the single state synthesis tool use the command:

% trans -auto < vhdl_file >

To invoke the multi state synthesis tool use the command:

% vss_ver5 [-architecture <architecture_type>] <input_file >

The <architecture_type> can be one of the following: (a) non_pipelined (b) status_pipelined (c) control_status_pipelined

For both these commands, a '.vhdl' extension is automatically assumed by VSS.

4.1.2 Commands for Results Display

The command for displaying textual results is:

% xdisp_file <results_filename>

The <results_filename> should be a valid textual filename. A list of files produced by VSS is shown in the table in Figure 4.

4.1.3 Flowgraph Display

The command for displaying the flowgraph is :

% xdp <flowgraph_filename>

A '.dgm' extension to the <flowgraph_filename> is automatically assumed. The <flowgraph_filename> should be a valid filename. A list of flowgraph filenames produced by VSS is shown in the table in Figure 4. Pressing the **<c>-key** inside xdp will provide a list of all possible flowgraph manipulation commands.

4.2 Interactive Mode

The Interactive Mode used can be used for complex VHDL descriptions with multiple blocks and processes. To invoke VSS in the interactive mode execute the command:

% VSS <vhdl_file>

A '.vhdl' extension to the filename is automatically assumed.

This command invokes the VHDL preprocessor which splits the given VHDL description into multiple entities with appropriate routing of the global signals.

After invoking the pre-processor a user interface (UI) like the one shown in Figure 5 appears on the screen. This UI can be used to invoke all synthesis commands, and also to view the results of the synthesis. The designer can use this UI to control the synthesis and a examine the results.

A block or process must be selected before invoking any command on the UI. A block can be selected by placing the cursor on any one of the boxes (representing VHDL blocks/processes) and clicking the first mouse key (m1). The selected block will immediately appear shaded. As an example, block *MAIN* in Figure 5 has been selected.

The menu-bar on the upper portion of the UI contains four buttons **File, Synthesis, Summary** and **Help**. Placing the cursor on any of these buttons and clicking on the first mouse key invokes the particular command related to the button. Invoking the 'File' and the 'Synthesis' commands will result in a submenu which contains a further list of commands. Some of the commands in the submenus may create further submenus.

In this document we use a simple notation to indicate commands that are available in the lower levels of the hierarchy of submenus. The command **Synthesis ⇒ Results ⇒ CU Netlist** can be accessed by first clicking on the **Synthesis** button in the menu bar. This would result in a submenu appearing on the screen. The cursor must be moved within the submenu to point to the **Results** item. Another submenu would appear, from which the **CU Netlist** option can be selected.

Let us now examine all the commands available from the User Interface. These commands are relevant to the block or the process that has been selected on the UI.

4.2.1 File ⇒ Quit

This command closes the User Interface and quits the program.

4.2.2 Synthesis ⇒ Read VHDL

As mentioned above, the preprocessors splits the input VHDL into smaller entities, with one entity per process or block. This command can be used to show the VHDL for the block or process selected in the UI.

4.2.3 Synthesis ⇒ Show Flowgraph

The flowgraph for the selected block or process can be seen with this command.

4.2.4 Synthesis ⇒ Execute VSS

This command invokes the synthesis process. The type of architecture that is produced depends on the *Control Style:* setting in the UI. (Section 4.2.14 explains how to set the architecture type.) This command produces the VHDL structure netlist for the datapath, a process description of the controller, and an overall netlist that combines the control unit and the datapath. It is possible to simulate the datapath netlist after this command.

4.2.5 Synthesis ⇒ Control Synthesis

This command invokes the control unit synthesis program. The control unit description produced during the datapath synthesis can be synthesized using this command. The output of this command is a netlist of gates, from the mcnc.genlib.

4.2.6 Synthesis ⇒ Show Results ⇒ Control Tables

This command shows the results of scheduling and binding. These results indicate the actions in each state and the activation of control pins on all datapath components. The format is very similar to the *Ops Based* and the *Unit Based* state tables in BIF [5].

4.2.7 Synthesis ⇒ Show Results ⇒ Overall Structure

This command shows the overall netlist consisting of two components (i) the control unit and the (ii) datapath. This file can be used to verify the number of pins on both the control unit and the datapath.

4.2.8 Synthesis ⇒ Show Results ⇒ Datapath Structure

This command shows the VHDL structural description of the datapath.

4.2.9 Synthesis ⇒ Show Results ⇒ ControlUnit Process

This command shows a VHDL process description of the control unit.

4.2.10 Synthesis ⇒ Show Results ⇒ ControlUnit Structure

This command shows the VHDL structural description of the control unit.

4.2.11 Summary

This command shows a summary of the components that have been used for synthesizing the description.

4.2.12 Help

This command shows any on-line help that may be available.

4.2.13 Control Style

The architecture type can be set by using the option menu that is available below the top menu bar. This command allows you to select one of the four possible architectures synthesizable by VSS. Proper setting of the architecture type is required before invoking the synthesis command.

This command works differently from the other commands discussed above. If we place the cursor on the rectangle to the right of the **Control Style** and click on the first mouse key (m1), all the possible choices for the architecture get displayed. Keeping the mouse key pressed, the cursor must be

moved to the desired architecture type. When the mouse key is finally released the desired architecture type will be selected and the selection will be displayed in the rectangle.

4.2.14 Datapath Style

This command determines the type of functional units that can be used (Pipelined or NonPipelined). Note that the setting of this command is not considered for the current version of the release.

5 Component Library

VSS uses a generic microarchitectural component library called GENUS. Since GENUS is a parameterized library, VSS can instatiate any component by specifying the required parameters (e.g., bit-width, functionality).

A generic library allows efficient synthesis by providing components with the required structure. In addition to providing the components, GENUS also generates VHDL models automatically. These VHDL models are created in the directory specified by the environmental variable **GENUS_VHDL_DIR**.

After the design is synthesized using generic components, each component can either be designed manually or an automatic component synthesis program like DTAS can be used. DTAS can synthesize microarchitectural components using the technology specific components from a vendor library.

References [6, 7] provide complete details about the GENUS library and the DTAS component synthesis program.

6 The UNIT_CONSTRAINT file

VSS accepts functional unit constraints during synthesis. It is possible to constrain the number of functional units of various types. A *UNIT_CONSTRAINT* file must be created in the same directory as the VHDL model.

In order to simplify the task of creating this file, we provide a sample *UNIT_CONSTRAINT* file in the release directory. Assuming that the model file is in the directory $HOME/models, the sample constraint file can be copied to the directory where the model exists.

% cp <release_dir>/vss_ver5/scripts/UNIT_CONSTRAINT $HOME/models

A part of a sample constraint file is shown in Figure 6.

The original file constrains the synthesis process to use three different functional units. Each functional unit definition is demarcated with a line containing a '#' sign. The first functional unit describes a 10 bit ALU, which performs the ADD and SUBTRACT function. The second component is another ALU that performs the COMPARISON and the NOT function. The third component is a 10 bit multiplier.

This file can now be edited to show the actual UNIT_CONSTRAINTS for the synthesis. Here are some of the ways the file can be modified.

- Changing the number of instances - The *NUM_INSTANCES* parameter can be changed to indicate the maximum number of allowed instances for that component. For example if a maximum of *two* multipliers are allowed, change the line *NUM_INSTANCES: 1* for the multiplier to *NUM_INSTANCES: 2.*

```
GC_COMPILER_NAME: ALU
GC_INPUT_WIDTH: 10
GC_NUM_FUNCTIONS: 2
GC_FUNCTION_LIST: GC_SUB,GC_ADD
GC_STYLE: GC_RIPPLE_CARRY
NUM_INSTANCES: 1
AREA: 100.0
DELAY: 80.0
#
GC_COMPILER_NAME: ALU
GC_INPUT_WIDTH: 16
GC_NUM_FUNCTIONS: 2
GC_FUNCTION_LIST: GC_EQ,GC_LNOT
GC_STYLE: GC_RIPPLE_CARRY
NUM_INSTANCES: 1
AREA: 100.0
DELAY: 80.0
#
GC_COMPILER_NAME: MULT
GC_INPUT_WIDTH: 10
GC_STYLE: GC_ARRAY
NUM_INSTANCES: 1
AREA: 3000.0
DELAY: 180.0
#
```

Figure 6: Part of a sample UNIT_CONSTRAINT

- Changing the bit width of components - The *GC_INPUT_WIDTH* reflects the bit width of the specified component. To change the bitwidth of the multiplier change the *GC_INPUT_WIDTH: 10* line for the multiplier to *GC_INPUT_WIDTH: 16*
- Deleting a component - A component can be deleted by changing the *NUM_INSTANCES* to 0.
- Creating a new type of component - The ALU in GENUS can perform many different functions. The possible functions are explained in the GENUS manual [6]. To create an ALU that performs the increment and decrement operations copy the nine lines (including the # line) that represent the ALU component, and change the GC_NUM_FUNCTIONS AND GC_FUNCTION_LIST parameters. In other words, the following lines would get appended to the UNIT_CONSTRAINT file.

```
GC_COMPILER_NAME: ALU
GC_INPUT_WIDTH: 16
GC_NUM_FUNCTIONS: 2
GC_FUNCTION_LIST: GC_INC,GC_DEC
GC_STYLE: GC_RIPPLE_CARRY
NUM_INSTANCES: 1
AREA: 100.0
DELAY: 80.0
#
```

- Changing Technology Specific Parameters - The AREA and the DELAY specification provides hints to the synthesis program. This can be changed to reflect reality. In the original constraint file the multiplier is twice as slow as the ALU.
- Note that the first two REGISTER components in the file must not be changed.

A program called **create_unit_constraint** is released with the package. This is an interface that provides a simple way of creating the UNIT CONSTRAINT file.

7 Tutorial

In this section we will walkthrough an example to understand how to use VSS and the type of designs that VSS can synthesize.

Let us synthesize a simple address calculation design. In this model, we increment the *pc* by 4 if the current instruction is not a branch instruction. Otherwise, it sets the *pc*equal to the input *branchpc*.

```
entity  addr_cal is
  port (branchpc : in bit_vector(15 downto 0);
        branch,clk : in bit;
        pc   : out bit_vector(15 downto 0) );
end addr_cal;
architecture behavior of addr_cal is
  variable pc_reg: bit_vector(15 downto 0);
begin
P1:  process (clk)
   begin
    if (clk = '1') then
       if (branch = '1') then
         pc_reg := branchpc;
       else
         pc_reg := pc_reg + 4;
       end if;
    end if;
    pc <= pc_reg;
   end process P1;
end behavior;
```

Figure 7: Addr_Calc Behavioral Description

Figure 7 shows the behavioral description of the circuit. It has a single process containing a simple branch statement. Let us go through the following steps for synthesizing this design.

1. Source the vss_script file
   % source <release_dir>/vss_ver5/bin/vss_script

2. Create a new working directory and move to this new directory:
   % mkdir $HOME/test_vss; cd $HOME/test_vss

3. Copy the VHDL file from the examples directory
   % cp <release_dir>/vss_ver5/examples/addr_calc/addr_calc.vhdl .

4. Copy the UNIT_CONSTRAINT file from the release directory:
   % cp <release_dir>/vss_ver5/examples/addr_calc/UNIT_CONSTRAINT
   .

5. Edit parts of the UNIT_CONSTRAINT to allow one 32 bit ALU for addition and one 1 bit ALU for comparison as shown below:

```
GC_COMPILER_NAME: ALU
```

```
GC_INPUT_WIDTH: 32
GC_NUM_FUNCTIONS: 1
GC_FUNCTION_LIST: GC_ADD
GC_STYLE: GC_RIPPLE_CARRY
NUM_INSTANCES: 1
AREA: 100.0
DELAY: 80.0
#
GC_COMPILER_NAME: ALU
GC_INPUT_WIDTH: 1
GC_NUM_FUNCTIONS: 1
GC_FUNCTION_LIST: GC_EQ
GC_STYLE: GC_RIPPLE_CARRY
NUM_INSTANCES: 1
AREA: 100.0
DELAY: 80.0
#
```

6. Invoke VSS on the description
   % VSS addr_calc
   The User Interface appears on the screen showing the overall structure of the chip.
7. Select the process P1, represented as a box on the UI. This box becomes highlighted.
8. Select the following command on the UI
   Synthesis ⇒ Show_Flowgraph
   The flowgraph for the process appears in a separate window.
9. Note that the **With Status Registers** architecture types has been selected by default.
10. Synthesize the circuit for the process by selecting the following command
    Synthesis ⇒ Execute_VSS
11. Synthesize the control logic by selecting the command:
    Synthesis ⇒ Control_Synthesis
12. Click on the Summary button. A form containing a summary of the components used by VSS will be displayed. A transistor count estimate is also provided.
13. Click on the 'Close' on the bottom of the summary form. The summary form will disappear.
14. Select any or all of the following commands to see the results of the synthesis
    (Synthesis ⇒ Show Results ⇒ Control Tables,
    Synthesis ⇒ Show Results ⇒ Overall Structure,
    Synthesis ⇒ Show Results ⇒ Datapath Structure,
    Synthesis ⇒ Show Results ⇒ ControlUnit Process,
    Synthesis ⇒ Show Results ⇒ ControlUnit Structure)

Other architectures can be synthesized by changing the architecture setting to the desired **Control Style** and repeating steps 10 to 14.

7.1 Simulation

All the VHDL files are created in a subdirectory (called RESULTS). Since the process is called P1, all the results files will be called V_P1_<something>...

All the VHDL models for the individual components will be created in the directory pointed to by the environmental variable **GENUS_VHDL_DIR**.

To simulate the complete design created by VSS, simulate the files in the following order.

- defs.vhd (This is available in $RELEASE_DIR/genus/defs.vhd)
- all files with a '.vhdl' extension in the directory pointed to by the environmental variable **GENUS_VHDL_DIR**.
- the datapath structure netlist - this is available in RESULTS/V_P1_dp_struct.vhdl
- the control unit process - this is available in RESULTS/V_P1_cu_process.vhdl
- the overall structure consisting of the CU and DP - this is available in RESULTS/V_P1_struct.vhdl

After compiling the files in the above order, test vectors can be applied to the overall structure.

8 Acknowledgements

This work was supported by the Semiconductor Research Corporation (grant #91-DJ-146). We are grateful for their support.

9 Appendix A

In this appendix we list some of the features of the VHDL language that are currently not supported by the VSS.

VSS does not support the following constructs in VHDL

- Procedure and Function Calls
- Enumerated Types
- Record Structures
- CONSTANT declarations
- Null statements
- Exit statements
- Return Satements
- loop with no iteration scheme
- Arrays of more than 2 dimensions

- Inout Ports

In addition to avoiding the above constructs, the following points must be observed, due to limitations in the current version of the software.

- Comment statements in the input VHDL cause problems to the VHDL preprocessor. They should be removed before the preprocessor is invoked.
- The preprocessor also requires that all processes and blocks have a name. This implies that every process should begin with
  NAME: process .....
  and end with
  end process NAME;

10 References

[1] J.Lis and D. Gajski, "Behavioral Synthesis from VHDL Using Structured Modeling." University of California, Irvine, TR ICS 91-05, 1991.

[2] J.Lis and D. Gajski, "Structured Modeling for VHDL Synthesis." University of California, Irvine, TR ICS 89-14, 1989.

[3] R.K.Brayton, A. V. R.Rudell, and A.Wang, "MIS: A multiple level logic optimization system," *IEEE Transactions on Computer-Aided Design*, Nov 1987.

[4] L.Ramachandran and D. Gajski, "Tradeoffs in Synthesis of Pipelined Controls." University of California, Irvine, TR ICS 92-49, 1992.

[5] T. N. Dutt and D. Gajski, "BIF: A Behavioral Intermediate Form for High Level Synthesis." University of California, Irvine, TR ICS 89-03, 1989.

[6] N. Dutt and P. Jha, "GENUS: A Generic Component Library for High Level Synthesis." University of California, Irvine, TR ICS (will be available in June 92), 1992.

[7] J. Kipps, "An Approach to Component Generation and Technology Adaptation." University of California, Irvine, TR ICS 91-79, 1991.

[8] "IEEE Standard VHDL Language Reference Manual.", 1988.