verilog2vhdl™ User's Manual

May 2007

SynaptiCAD, Inc.,
PO Box 10608
Blacksburg, VA 24062-0608
Ph: (540) 953-3390 Fax: (540) 953-3078
email: sales@syncad.com

Copyright © 2007, Alternative System Concepts, Inc. All rights reserved worldwide.
May be printed or photocopied by licensed customers of ASC for internal business purposes only.

Table of Contents

- About this manual
  - Conventions
  - References

Chapter 1: Introduction and Installation

- Introduction

Chapter 2: Verilog2VHDL Translation Overview

- Key Concepts
- Verilog to VHDL Translation
  - Objects and Types
  - Module
  - Parameter Declaration
  - Register and Net Declaration
  - Memory Declaration
  - Integer Declaration
  - Real Declaration
  - Module Instantiation
  - Gate Instantiation
  - Always Statement
  - Initial Statement
  - Conditional 'if-else-if' Statement

    - Conditional 'case' Statement
    - Conditional 'casex' and 'casez' Statement
    - Looping Statements
    - Continuous Assignments
    - Procedural Assignments
    - Non-blocking Procedural Assignment
    - Compiler Directives
    - User-defined Tasks and Functions
    - System Tasks and Functions
- verilog2vhdl Error Handling
- Reserved words in verilog2vhd

Chapter 3: Using verilog2vhdl

- Command Summary
- When you have problems...

Chapter 4: Compiling and simulating output VHDL

- Order of Compilation for verilog2vhdl Packages
- RTL Translation Example
    - Input file
    - verilog2vhdl Invocation and Transcript
    - Output file
    - VHDL Compilation
- Gates and Parameteric Delay Example
    - Input File
    - verilog2vhdl Invocation and Transcript
    - Output File
    - VHDL Compilation
- '-Package' switch usage Example
    - Input File
    - Input File 2
    - verilog2vhdl Invocation and Transcript
    - Output File
    - Output File 2
    - VHDL compilation

About this Manual

Conventions | References ||| Back to Table of Contents

This manual describes how to install and use **verilog2vhdl**, a translator that produces VHDL design descriptions from Verilog HDL design descriptions.

Chapter 1 gives a brief overview of **verilog2vhdl** and explains how to install it. Chapter 2 gives an overview of how **verilog2vhdl** works and details how it translates Verilog elements to VHDL. Chapter 3 explains the command options you can use to fine tune the tool's operation.

You should use this Manual in conjunction with the latest **verilog2vhdl Release Notes** document, which provides the latest information on language support and usage details.

General questions and inquiries can be sent to:

SynaptiCAD, Inc.
PO Box 10608
Blacksburg, VA 24062-0608
Ph: (540) 953-3390 Fax: (540) 953-3078
email: sales@syncad.com

Bug reports and other support questions can be sent to:

email: sales@syncad.com

Conventions

In this manual, *italic text* is used to differentiate file and pathnames. **Bold print** is used for commands. `Courier type` is tool output or HDL code. **`Bold Courier type`** is text entered by the user on the command line.

References

The **verilog2vhdl Release Notes** that came with your version of the tool provide up-to-the-minute details on language support and execution command options. You can also find examples and other helpful information at the SynaptiCAD Web site.

These general references may also prove helpful as you work with Verilog and VHDL:

- IEEE Standard VHDL Language Reference Manual
- Verilog hardware Description Language Reference Manual (LRM), Version 2.0
- VHDL Designer's Reference, Jean-Michel Bergé, Alain Fonkoua, Serge Maginot and Jacques Rouillard
- The Verilog Hardware Description Language, D.E. Thomas and Philip Moorby

Chapter 1: Introduction

Back to Table of Contents

verilog2vhdl translates hierarchical Verilog HDL to VHDL by using an HDL object kernel. The present form of the tool performs translation of all structural and RTLVerilog, as well as a large subset of behavioral Verilog. It uses a combination of IEEE and tool-specific VHDL packages to perform the translation. Either one of IEEE 1076-1987 or IEEE 1076-1993 compliant VHDL can be created by the tool. Any VHDL produced by this system is compatible with a VHDL simulator that accepts IEEE 1076 VHDL.

verilog2vhdl can

- be useful to designers working in a Verilog/VHDL environment,
- be integrated into VHDL-only based tools,
- provide library translation capability, and
- automate the model generation procedure.

If you need to derive or support designs in both VHDL and Verilog, what do you do?

An environment that supports co-existence of Verilog HDL and VHDL designs is shown below.

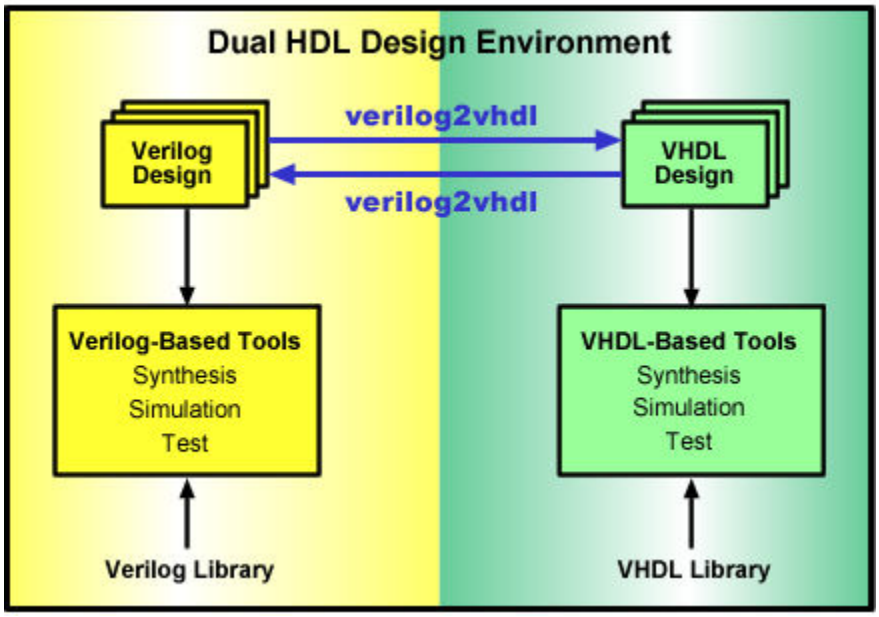

Figure 1: verilog2vhdl design flow

verilog2vhdl is a Verilog to VHDL translator, aiding you in moving your Verilog designs to VHDL. VHDL2verilog is the VHDL to Verilog translator. Using these translators, you can integrate your designs in the preferred synthesis-simulation-test environment.

The need to translate Verilog designs may arise due to varied circumstances. The design flow below will hold in most cases:

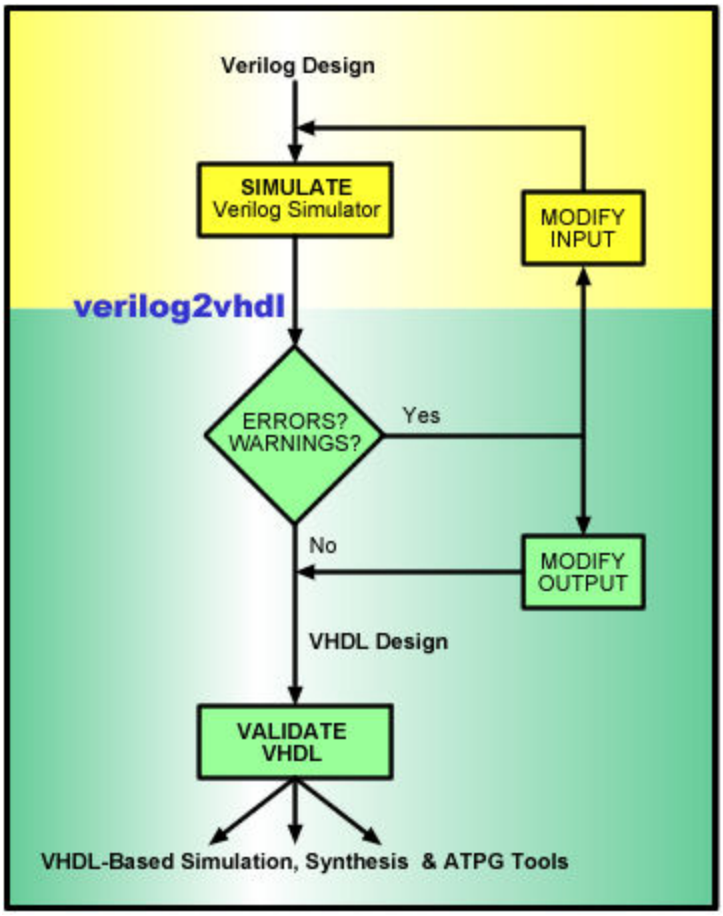

Figure 2: verilog2vhdl design flow

Figure 3 and Figure 4 show methods for validating your output VHDL design.

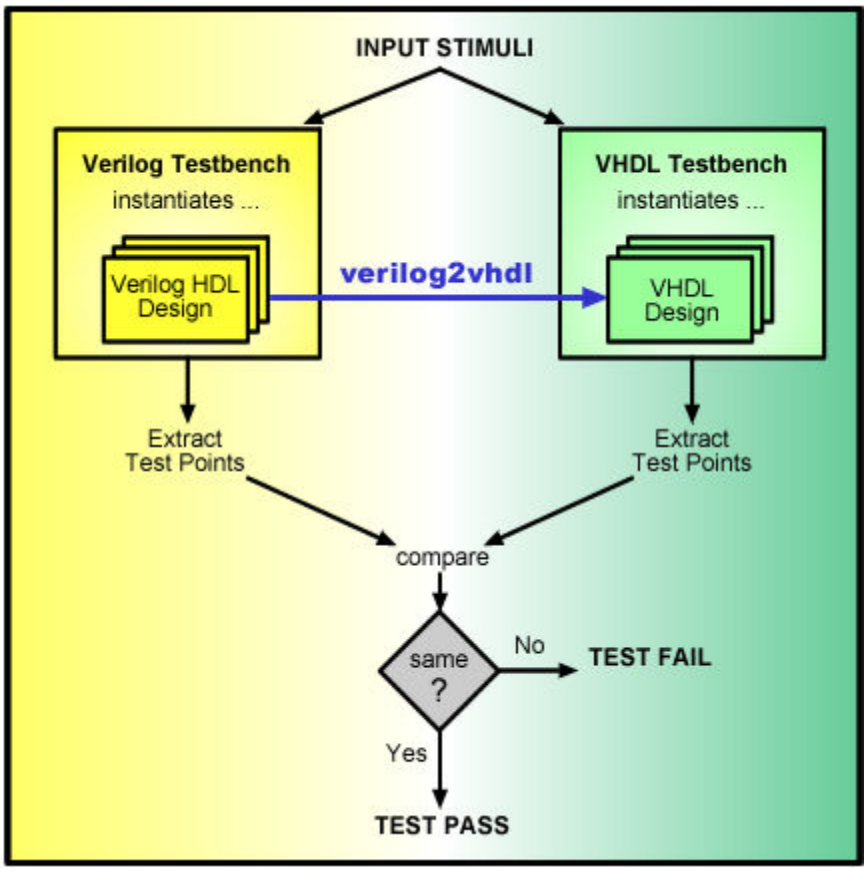

Figure 3: verilog2vhdl validation methodology - 1

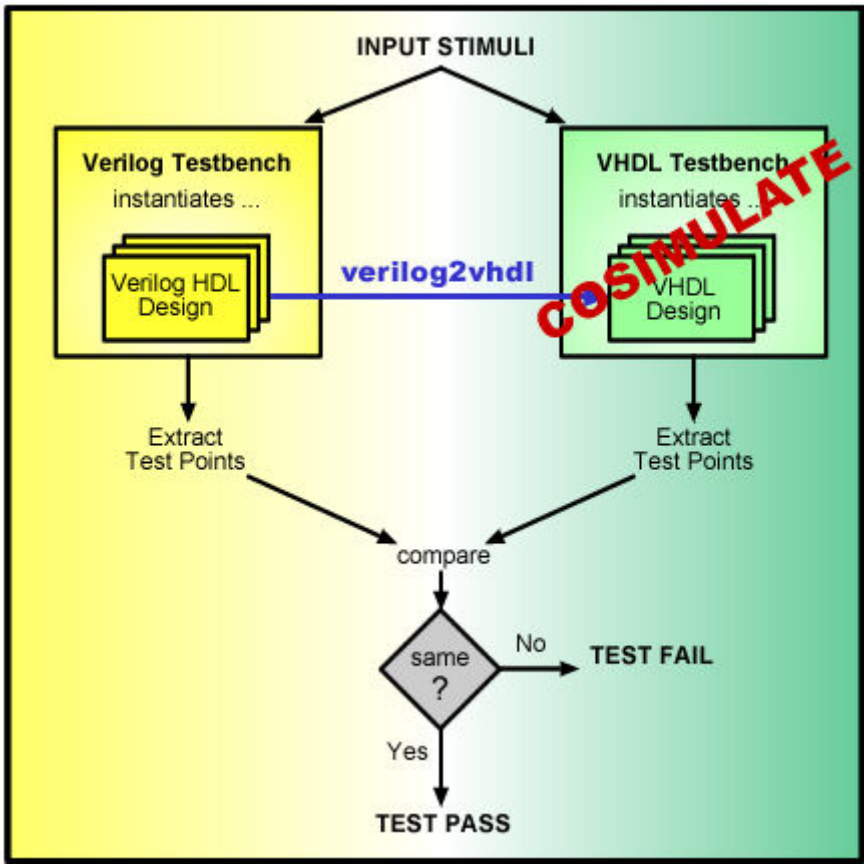

Figure 4: verilog2vhdl validation methodology - 2

The general goal of validation is to ensure that the output VHDL translation is functionally equivalent to the original Verilog design.

To achieve this objective, the same testbenches can be used to provide stimulus for both Verilog and VHDL designs, and output lists compared for equivalence. If a Verilog testbench already exists, the corresponding VHDL testbench can be created with the aid of **verilog2vhdl**. Alternatively, if you have access to a Verilog-VHDL co-simulator, you can co-simulate the existing Verilog testbench and the new VHDL design.

Chapter 2:
verilog2VHDL Translation Overview

Key Concepts | Verilog to VHDL Translation | Error Handling | Reserved Words ||| Back to Table of Contents

Key Concepts

verilog2vhdl translates Verilog to VHDL using a combination of IEEE and tool-specific packages. The tool provides support for comprehensive modeling styles and maintains hierarchy during the translation. In this chapter, various supported constructs are discussed.

The rest of the chapter is organized as follows: For every construct supported by **verilog2vhdl**, the general Verilog and VHDL semantics are given, followed by an example illustrating the construct in both languages. Special notes are given when required. The user is advised to refer to the VHDL LRM and Verilog LRM for detailed explanations.

Verilog to VHDL Translation

Objects and Types | Module | Parameter Declaration | Register and Net Declaration | Memory Declaration | Integer Declaration | Real Declaration | Module Instantiation | Gate Instantiation | Always Statement | Initial Statement | Conditional 'if-else-if' Statement | Conditional 'case' Statement | Conditional 'casex' and 'casez' Statement | Looping Statement | Continuous Assignments | Procedural Assignments | Non-blocking Procedural Assignments | Compiler Directives | User-Defined Tasks and Functions | System Tasks and Functions ||| Chapter Start

Objects and Types

Section Start

Verilog has the following types defined:

Register
Net
Integer, real and time variables

In VHDL, three classes of objects (as outlined below) are defined, each of which can be a scalar, composite, access or file type:

Signal
Variable
Constant

verilog2vhdl translates registers, integer, real and wire variables (henceforth referred to as Verilog objects) declared in Verilog to VHDL objects of a specific class based on the following criteria:

1. If a Verilog object is declared inside a task or function, the corresponding VHDL object class is VARIABLE.
2. If a Verilog object is declared as a parameter, the VHDL object class is CONSTANT.
3. All others Verilog object declarations get the VHDL object class SIGNAL.

Verilog and VHDL are based on the same 4-level logic system (0,1,X,Z). **verilog2vhdl** uses the IEEE standard 9-level std_ulogic type to translate the logic-levels. The logic levels in Verilog translate to the VHDL std_logic_1164 package 9-level system in the following way:

Logic Level Mapping

Verilog	VHDL
0	0
1	1
X	X
Z	Z
	U, H, L, -

The basic Verilog data type wire is equivalent to the VHDL resolved type std_logic. Similarly, a vector in Verilog is equivalent to the VHDL resolved type std_logic_vector.

Module

Section Start

The primary design unit in Verilog is the **module**. The equivalent representation in VHDL is an entity and architecture declaration. The standard Verilog and VHDL syntax for design unit declaration and an example illustrating the concept are shown below:

Verilog

```
module name_of_module (list_of_ports);

// input declaration
// output declaration
// inout declaration


ent_name is
// net declaration
// register declaration
```

VHDL

```
entity ent_name is

-- in port declaration
-- out port declaration
-- inout port declaration
end entity;

architecture arch_name of

--  signal declaration
--  signal declaration
```

```
// always statement

endmodule
```

Example:

```
module verilog2vhdl_example
verilog2vhdl_example is
      (verilog2vhdl_a,
       verilog2vhdl_b,
       verilog2vhdl_c);

input verilog2vhdl_a;
std_logic;
input verilog2vhdl_b;
std_logic;
output verilog2vhdl_c;
std_logic);



verilog2vhdl_example is

endmodule
```

```
begin
--  process statement

end arch_name

entity




port (verilog2vhdl_a : in

      verilog2vhdl_b : in

      verilog2vhdl_c : out

end entity;

architecture VeriArch of

  begin
end VeriArch;
```

Parameter Declaration

Section Start

Constants can be declared in Verilog using the **parameter** declaration. The corresponding VHDL declaration is a generic in the entity declaration. The Verilog and VHDL syntax and an example follow below.

Verilog

```
........

parameter assignment

.........
```

Example:

```
...........

parameter a = 10;
parameter b = 5;
parameter c = 1.5;

........
```

VHDL

```
entity name_of_entity is

-- generic_interface_list

end entity;

..........

generic (CONSTANT a : integer := 10;
      CONSTANT b : integer := 5;
        CONSTANT c : real := 1.5);

...........
```

Register and Net Declaration

Section Start

There are two main types of data types in Verilog: Registers and Nets. Registers are a data type to model data storage. They can hold their value until the next assignment. Nets, on the contrary, do not store any value (except trireg), and instead take the value of the driving gate or assignment. They are primarily used to represent connections between structural entities. VHDL models both registers and nets using the type **signal**. Signals in VHDL are used both for electrical connections and storing values.

Verilog

```
.............

name_of_entity is
net_declaration;
reg_declaration;

.............
```

Example:

```
.........

wire a;
wire [3:0] b;
downto 0);
reg c;

............
```

VHDL

```
entity name_of_entity is
end entity;

architecture arch_name of

      signal_declaration;
      signal_declaration;

end architecture;

.........

       signal a: std_logic;
       signal b: std_logic_vector(3

       signal c: std_logic register;

.............
```

Special Note: Register declarations and other Verilog declarations inside user defined tasks and functions are modeled as VHDL **variables**.

Memory Declaration

Section Start

Verilog HDL models memories as arrays of register variables. They are typically used to model Read only memories (ROM) or Random access memories (RAM). Translation of memory types to VHDL is straight forward. A special type is created for this purpose in output VHDL.

Verilog

```
.............

is
memory_declaration;

.............
```

Example:

```
.........
```

VHDL

```
entity name_of_entity is
end entity;

architecture arch_name of name_of_entity

   memory type_declaration;

end architecture;

.........
```

```
reg [255:0] a[7:0];
```

```
type memory_0 is array (7 downto 0) of
  std_logic_vector (255 downto 0);
signal a : memory_0 register;
```

```
............
```

```
............
```

Integer Declaration

Section Start

One of the types allowed in Verilog is the Integer type. Integer types in Verilog are translated to a resolved subtype of the pre-defined integer type in VHDL.

Verilog

```
..............
```

```
integer_declaration;
integer_array_declaration;
```

```
..............
```

Example:

```
..........
```

```
integer a;
integer b[3:0];
```

```
............
```

VHDL

```
entity name_of_entity is
end entity;
```

```
architecture arch_name of name_of_entity
is
    signal_declaration;
    signal_declaration;
```

```
end architecture;
```

```
use Verilog.v2v_types.all;
.........
```

```
   signal a: v2v_integer REGISTER;
   signal b: integer_array(3 downto 0);
```

```
.............
```

Special Note: The resolved subtype **v2v_integer**, and type **integer_array** are declared in the **v2v_types** package available in the *types.vhd* file.

Real Declaration

Section Start

Types allowed in Verilog include the **real** type. Real types in Verilog are translated to the pre-defined **real** type in VHDL.

Verilog

```
..............
```

```
name_of_entity is
real_declaration;
```

```
..............
```

Example:

VHDL

```
entity name_of_entity is
end entity;
```

```
architecture arch_name of
```

```
   signal_declaration;
```

```
end architecture;
```

```
                                    use Verilog.v2v_types.all;
.........                           .........

real a;                                signal a: real;

...........                         ............
```

Module Instantiation

Section Start

Both Verilog and VHDL are hierarchical description languages, and allow modules to be embedded inside other modules. This process is also known as module instantiation. Higher level modules instantiate modules, and communicate with them through the ports on the instantiated module. The functionality of module instantiation is equivalent in both languages.

Verilog

```
...........
entity_name is

module_instance_name module_name
embedded_entity_name
list_of_module_connections

.........
```

Example:

```
...........
entity_name is
wire a,b;
reg c;

df_instance1 df (a,b,c);
(a,b,c);

df_instance2 df (.in1(a),
(in1=>a,
             .in2(b),
in2=>b,
             .out1(c));
out1=>c);
............
```

VHDL

```
architecture arch_name of

begin
   instance_name

   port_association_list

end architecture;

architecture arch_name of

    signal a,b,c : std_logic;

begin
    df_instance1 df port map


    df_instance2 df port map




end architecture;
```

Gate Instantiation

Section Start

Verilog is a rich language for modeling at low levels of abstraction. It has built-in primitives like n-input nand gates and buffers. VHDL, on the other hand, is more suited for modeling at higher levels. Though it is possible to create entities corresponding to

Verilog gates, **verilog2vhdl** translates a gate instantiation for almost all gates into equivalent VHDL concurrent signal assignments.

Verilog

```
.....
is

// wire_declaration;
// gate_instantiation;

.....
```

Example:

```
.............
`timescale 1ns/1ns



wire a,b,c;
wire d,e,f;
wire g,h,i;

nand (a,b,c);
nor  #3 (d,b,c,e,f);
after 3 ns;

not  #2 (g,b);
not (h,i,c);

.....
```

VHDL

```
architecture arch_name of entity_name

begin

-- concurrent signal assignment;

end architecture;

.....
```

```
  signal a,b,c : std_logic;
  signal d,e,f : std_logic;
  signal g,h,i : std_logic;

  a <= b nand c;
  d <= (((b nor c) nor e) nor f)


  g <= b after 2 ns;
  h <= c;
  i <= c;
.....
```

Always Statement

Section Start

Verilog

```
.........

always
begin

// event_control statement

end

........
```

Example:

```
.............
reg b;
wire a;
always @ (negedge clock)
begin
   if (a == 1)
      b = 1;
```

VHDL

```
...........

process
begin

-- event_control statement

end process;

...........
```

Example:

```
USE Verilog.timing.all;
  ..............
  signal a,b : std_logic;
 ...........
  process
  begin
     wait until negedge(clock);
     if (a = '1')
```

```
  else
        b = 0;
end
```

```
        b = '1';
    else
        b = '0';
    end if;
end process;
```

```
...........
```

```
.............
```

Special Note: `negedge' is a library function provided in the **timing** package (file timing.*vhd*).

Initial Statement

Section Start

The initial statement is similar to the always statement, except that it executes only once during the entire simulation run. It is functionally equivalent to a VHDL process with an infinite wait statement at the end of the sequential body.

Register initialization in initial statements without event control are handled as a special case. If the register initialization is the first statement in the initial block without event control (precedes ALL wait statements), the statement is mapped to a signal initialization statement in VHDL. The following example shows this special case for register `a'.

Verilog

```
.........
initial
begin

// optional event_control statement
statement

end
........
```

VHDL

```
...........
process
begin

  -- optional event_control


end process;
...........
```

Example:

```
..............
reg a;
`0';

initial
begin
  a = 0;
  #10 b = 0;
  c = 1;
  d = 1;
end

.............
```

```
........
SIGNAL a : std_logic REGISTER :=

..........
process
begin
   wait for 10 ns;
   b <= '0';
   c <= '1';
   d <= '1';
   wait;
end process;
.........
```

Conditional 'if-else-if' Statement

Section Start

Both languages support the conditional `if statement' and the translation is mostly straightforward, with a few exceptions. In Verilog, the result of an `if expression' can be (0,1,X,Z). In VHDL, the `if expression' has to decode to the boolean type in VHDL (FALSE, TRUE). All Verilog expressions do not map directly to VHDL. A typical example is the following `if expression' (this type of expression will be referred to a `non-boolean' expression in ensuing discussion) :

```
      if (a)
```

There is no direct equivalent in VHDL, because the above `if expression' tests if `a' is `1', in the case `a' is a register data type, or if `a' has a value other than `0', if `a' is an integer type. To preserve logical equality during translation, **verilog2vhdl** translates a non-boolean `if expression' in Verilog to a boolean expression. A negation test is applied to the scalar or vector instead, to preserve the logic. An example illustrates the translation below. In the example, the first `if expression' is not a boolean expression by itself. Hence, **verilog2vhdl** maps the `if expression' in Verilog to VHDL by first applying the std_logic_1164 package translation function `To_Bit' to the register variable and then testing for a non-zero value. The translation function `To_Bit' is required to discard cases involving unknowns.

Example:

Verilog

```
........
always
begin

// conditional_statement

end
........
.........

always @a
begin
if (a)
      b = 1;
else
      b = 0;
end


........
```

VHDL

```
.........
process
begin

  -- conditional_statement

end process;
.........
.........

process
begin
   wait on a;
   if To_bit(a)  /= '0' then
       b <= '1';
   else
       b <= '0';
   end if;
end process;
.........
```

Special Note: The library function `To_bit' is available in the **Std_logic_1164** package.

Conditional 'case' Statement

Section Start

The case statement uses the same reasoning in both languages; both are multi-way decision statements which test for a matching expression and branch accordingly. There are, however, some limitations in VHDL, which does not have direct equivalent

statements for `casex' and `casez'. The translation of `casex' and `casez' statements is detailed in the next section.

Verilog

```
.........
// Sequential process
// case statement
.........
```

VHDL

```
.........
   -- process statement
   -- case statement
.........
```

Example:

```
.........
always @(select)
begin
case (select)
   3'b 000 : dec_out = 8'b 00000001;
   3'b 001 : dec_out = 8'b 00000010;
<= "00000001";
   3'b 010 : dec_out = 8'b 00000100;
<= "00000010";
   3'b 011 : dec_out = 8'b 00001000;
<= "00000100";
   3'b 100 : dec_out = 8'b 00010000;
<= "00001000";
   3'b 101 : dec_out = 8'b 00100001;
<= "00010000";
   3'b 110 : dec_out = 8'b 01000001;
<= "00100000";
   3'b 111 : dec_out = 8'b 10000001;
<= "01000000";
endcase
<= "10000000";
end
.........
```

```
.........
process
begin
  wait on select;
         case select is
            when "000" => dec_out

            when "001" => dec_out

            when "010" => dec_out

            when "011" => dec_out

            when "100" => dec_out

            when "101" => dec_out

            when "110" => dec_out

            when "111" => dec_out

          end case;
end process;
...........
```

Conditional 'casex' and 'casez ' Statement

Section Start

`casex' and `casez' statements are special purpose case routines provided in the Verilog language for `dontcare' comparison. VHDL has no direct equivalent, but **verilog2vhdl** writes out the equivalent VHDL for a `casex' or `casez' statement. Each of these statements is translated to an equivalent VHDL `if' statement and the sequential statements are updated accordingly.

Verilog

```
.........
// Sequential process
// casex statement
// casez statement
.........
```

VHDL

```
.........
-- process statement
-- if statement
-- if statement
.........
```

Example:

```
.........
always @(select)
begin
casex (select)
   3'b 0?0 : dec_out = 8'b 00000001;

   3'b ?01 : dec_out = 8'b 00000010;
then

   default : dec_out = 8'b Z;

endcase
end

always @(select)
begin
casez (select)
   0 : dec_out = 8'b 00000001;

   1 : dec_out = 8'b 00000010;

  default : dec_out = 8'b Z;

endcase
end
........
```

```
..........
process
begin
      wait on select;
      if casex(select,"0Z0") then
        dec_out <= "00000001";
      elsif casex(select,"Z01")

        dec_out <= "00000010";
      else
        dec_out <= "ZZZZZZZZ";
     end if;
 end process;

 process
 begin
      wait on select;
      if casez(select,0) then
        dec_out <= "00000001";
      elsif casez(select,1) then
        dec_out <= "00000010";
      else
        dec_out <= "ZZZZZZZZ";
      end if;
  end process;
  ...........
```

Special Note: `casex' and `casez' are library functions available in the **utils** package (file *utils.vhd*).

Looping Statements

Section Start

Looping, or iteration schemes found in Verilog can be easily translated to VHDL. **verilog2vhdl** supports `for', `repeat', `while' and `forever' statements.

Verilog

```
.......

for (initial; end; update)
      statement;



while (expr)
      statement;


repeat (expr)
loop
      statement;


forever
```

VHDL

```
...........
initial;
   while (end) loop
   statement;
   update;
   end loop;

   while (expr) loop
   statement
   end loop;

   for identifier in expr downto 1

   statement;
   end loop;

   loop
```

```
      statement;                          statement;
                                          end loop;

........                                ..........
```

Example:

```
........                                ..........
                                        i := 0;
for (i = 0; i < 4; i = i + 1)             while (i < 4) loop
      b = b + 2;                            b := b + 1;
                                            i := i+1;
                                          end loop;

while (i < 5)                             while (i<5) loop
      b = b + 2;                            b := b + 2;
                                          end loop;

repeat (5)                                for V2V_repeat in 5 downto 1
loop
      b = b + 2;                            b := b + 2;
                                          end loop;

forever                                   loop
      b = b + 2;                            b := b + 2;
                                          end loop;

........                                ..........
```

Special Note: The Verilog `for' statement is equivalent to the VHDL `for' statement only when the Verilog `for' has static bounds, and the loop variable (`i' in the example above) is incremented by one (i = i + 1) or decremented by one (i = i - 1). The loop variable `i' is not available as a **signal** in VHDL, and hence cannot be accessed when used as the target for assignments, other than incrementing (decrementing) by one. For this reason, the Verilog `for' is mapped to the VHDL `WHILE' loop. In doing so, we also gain access to the loop variable `i'.

Continuous Assignments

Section Start

A continuous assignment in Verilog is a statement that executes every time the right hand side of the an assignment changes. This is wholly equivalent to the concurrent signal assignment in VHDL with inertial delays.

Verilog **VHDL**

```
........                                .........

// continuous assignment                -- concurrent signal assignment

.........                               .........
```

Example:

```
..........                              .........

`timescale 1ns/1ns
```

```
assign #10 a = b & c;                  a <= b and c after 10 ns;

..........                             ...........
```

Procedural Assignments

Section Start

Procedural assignments are used to assign values to register, integers, real or time variables. These types of assignments are normally present in always/initial statements, tasks and functions. Procedural assignments can have delay, event or repeat control. All cases are translated correctly to functionally equivalent VHDL.The subtle differences when translating a Verilog procedural assignment to a VHDL assignment are shown below.

Example:

```
Verilog                                    VHDL

............                               ..................

// initial statement                       -- sequential body
// procedural assignment                   -- signal assignment

...........                                ......................
`timescale 1ns/1ps;

reg a,b;                                   signal a, b : std_logic;
                                           shared variable V2V_a : std_logic;
always @(clk)                               process
begin                                       begin
                                               wait on clk;
      a = #10 1;                               a <=`1' after 10 ns;
                                               wait for 10 ns;

      a = @(posedge clk) b;                    V2V_a := b;
                                               wait until posedge(clk);
                                               a <= V2V_a;

      a = repeat (5) @(posedge clk) b;         V2V_a := b;
                                               for V2V_repeat in 5 downto 1
loop
                                                 wait until posedge(clk);
                                               end loop;
                                               a <= V2V_a;


                                               wait for 5 ns;
      #5 b <= 0;                               b <= transport `0';

      #5 b = 1;                                wait for 5 ns;
                                               b  <= `1';
                                               wait for 0 ns;


end                                             end process;
```

```
.................                    ..................
```

Special Note: Explanation follows about the `wait for 0 ns' statements in the VHDL code corresponding to a Verilog blocking assignment without timing control. In Verilog, events due to blocking assignments always occur. Statements following the blocking assignments occur only after the blocking assignment has taken effect. In VHDL, however, simulation time advances only after a wait statement, and assignments take effect when simulation time advances. For this reason, it is necessary to insert a `wait for 0 ns' after a sequential signal assignment in VHDL for cases where no delay in specified.

Non-blocking Procedural Assignment

Section Start

The non-blocking Verilog procedural is a way to model transport delays in signal assignments; i.e it does not matter what order you make the assignments in a procedural dataflow. VHDL assignments are inherently non-blocking, and hence it is easy to translate Verilog non-blocking assignments.

Example:

Verilog

```
.............

// initial statement
// procedural assignment

...........

`timescale 1ns/1ps;
.............

always @(posedge clk)
begin

  a <= #10 1;

  #5 b <= 0;
end

.............
```

VHDL

```
.................

-- sequential body
-- signal assignment

.....................

.................

process
begin
  wait until posedge(clk);
  a <= transport `1' after 10 ns;
  wait for 5 ns;
  b <= transport `0';
end process;

.................
```

Special Note: `posedge' function is available in the **timing** package.

Compiler Directives

Section Start

Verilog supports numerous compiler directives. In this version of **verilog2vhdl**, only the `timescale and `define compiler directives are supported. `define compiler directives are translated as system - wide generic declarations i.e. all modules following the `define statement have a generic interface list corresponding to the `define statement. The time unit specified with the `timescale directive is used as the time unit for VHDL.

Verilog

```
.....................

// define compiler directive
declaration
// timescale compiler directive
description
.....................
```

Example:

```
`define TRUE 1
:= 1;
`define FALSE 0
:= 0;
`define foo "some people"

"some people";
`define all_zs 8'b Z

downto 0) :=


`timescale 10ns/1ns
reg a,b;



always @(posedge clk)
begin

      a = #10 1;

      #5 b = 0;
end
.............
```

VHDL

```
.........................

-- generic in entity

-- timeunit in VHDL

.........................

entity verilog2vhdl is port
   ( constant TRUE : integer

     constant FALSE : integer

     constant foo :
       string (1 to 10) :=

     constant all_zs :
       std_logic_vector(8

        "ZZZZZZZZ");


signal a,b : std_logic;



process
begin
wait until posedge(clk);
  a <= `1' after 100 ns;
  wait for 50 ns;
  b <= `0';
end process;
.............
```

IF support of other compiler directives is required, the user can use the **vpp** Verilog Pre-Processor, which can be found in the $ASC_HOME/v2v/bin directory. This pre-processor can handle all common compiler directives (i.e. the non-simulator specific ones).

Usage: **vpp *input_verilog_file output_verilog_file***

If the output_file is not specified, output is printed to `stdout`. This output file can then be the input to **verilog2vhdl**.

User-Defined Tasks and Functions

Section Start

User-defined tasks and functions are used extensively in behavioral Verilog. They encapsulate blocks of sequential statements and are invoked from within the module.

Additionally, arguments can be passed and exchanged through these calls. Though VHDL has equivalent procedures known as `Subprograms', they differ from Verilog tasks and functions in the following ways:

1. To execute a signal assignment in a VHDL subprogram, it is necessary for the signal to be available in the list of interface elements of the subprogram. This is not a requirement in Verilog. Hence, **verilog2vhdl** adds signals that are driven from within a Verilog task or function as interface elements to the corresponding VHDL subprogram.

2. In VHDL, it is not possible to read a signal inside a subprogram if it is an interface element of the subprogram of mode `OUT'. For this reason, an intermediate temporary signal is created in the VHDL architecture.

The above points are further illustrated by the example below.

Example:

Verilog

```
........
// task declaration
// function declaration
.........
..........
`timescale 1ns/1ns
module dummy(a,b,c);



REGISTER;

REGISTER) IS









`V2V_TEMP_temp'


input a;
input b;
output c;


reg temp;

always @(a)
      update_output;
```

VHDL

```
.........
 -- VHDL procedure
 -- VHDL function
.........

.........

ENTITY ... END ENTITY;
ARCHITECTURE VeriArch of dummy IS
PROCEDURE update_output(
  SIGNAL temp : OUT std_logic

  SIGNAL V2V_c : OUT std_logic

BEGIN
  temp <= a OR b;
  WAIT FOR 0 NS;
  V2V_c = V2V_TEMP_temp;
  -- note the temporary signal

 END PROCEDURE;

SIGNAL temp : std_logic;
SIGNAL V2V_c : std_logic;
SIGNAL V2V_TEMP_temp : std_logic;
SIGNAL GUARD : Boolean := true;
BEGIN
  PROCESS
  BEGIN
    WAIT on a;
    update_output(temp,c);
  END PROCESS;
```

```
task update_output;
temp = a or b;
c = temp;
endtask
above
endmodule

.........
```

```
  c <= GUARDED V2V_c;
  V2V_TEMP_temp <= GUARDED temp;
  -- Note the concurrent assignments

END VeriArch;

..........
```

System Tasks and Functions

Section Start

There are no direct VHDL equivalents to Verilog system tasks or functions, so the translation is done by automatically creating equivalent procedures in the VHDL output file. Of all Verilog system tasks and functions, **verilog2vhdl** supports those associated with formatted output, and $time and $fopen functions.

SYSTEM FUNCTIONS	
Verilog	**VHDL**
$time	NOW / <timeunit>
$fopen("filename.ext")	 FILE F1: text open WRITE_MODE is "filename.ext"

$time function is translated into NOW function normalized by 1 ns (default) or by the timeunit specified with a `timescale directive.

Translation of $fopen("filename.ext") function results in a declaration of a file with the logical name "filename.ext". If $fopen function appears on the RHS of an assignment or as an argument to a task or a function, it is substituted by the channel descriptor number normally returned by a $fopen function call in Verilog (see assignment to file1 in the example below). As in Verilog, filechannel 1 is reserved for STDOUT ("*/dev/tty*" in VHDL). STDOUT channel is added automatically to the translation of a Verilog file with output system task calls.

SYSTEM TASKS	
Verilog	**VHDL**
$display $fdisplay $strobe $fstrobe	V2V_display
$write $fwrite	V2V_write

The number of parameters of V2V_display is adjusted automatically to translate $(f)display or $(f)strobe call with the greatest number of parameters. Similarly, the number of parameters of V2V_write is adjusted automatically to translate the $(f)write call with the greatest number of parameters.

Verilog

```
// $time

// $fopen

// output system task call

.....................
```

VHDL

```
-- NOW / <timeunit>

-- file declaration
-- filechannel parameter assignment

-- declaration of string constants
-- V2V_display or V2V_write procedure
-- V2V_display or V2V_write call

.........................
```

Example:

```
integer file1;
integer filechan;


"latch.list";

"/dev/tty";

:= " ";

15) :=


11) :=


...

file1 = $fopen("latch.list");
filechan = file1 | 1;
$fdisplay(filechan, , $time,
v2v_message0, OPEN,
    "Change in qOut=%b with data=%b",
NS), OPEN,
     qOut, data);
to_bitvector(qOut),

to_bitvector(data));
```

```
SIGNAL file1 : integer;
SIGNAL filechan : integer;
SIGNAL std_io : integer := 1;
FILE F2 : text open WRITE_MODE is

FILE F1 : text open WRITE_MODE is

CONSTANT v2v_message0 : string (1 TO 1)

CONSTANT v2v_message1 : string (1 TO

                "Change in qOut=";
CONSTANT v2v_message2 : string (1 TO

                " with data=";
PROCEDURE V2V_display
    (SIGNAL filechan : in integer;
     message1 : IN string := "";
     signal1 : IN bit_vector := "";
     message2 : IN string := "";
     signal2 : IN bit_vector := "";
     message3 : IN string := "";
     signal3 : IN bit_vector := "";
     message4 : IN string := "";
     signal4 : IN bit_vector := "";
     message5 : IN string := "");
 ...

     file1 <= 2;
     filechan <= file1 OR 1;
     V2V_display(filechan,
                 INTEGER'IMAGE(NOW/1
                 v2v_message1,
                 v2v_message2,
```

verilog2vhdl Error Handling

Chapter Start

verilog2vhdl handles errors in the following way:

1. If a syntax error in encountered in the Verilog input file, the tool exits immediately after printing out the error message.

2. If an unsupported construct is encountered in the Verilog Input, a warning message is issued, and **verilog2vhdl** continues processing.

3. If the unsupported construct is of the type `event', **verilog2vhdl** exits after issuing appropriate error message.

Reserved words in verilog2vhdl

Chapter Start

It is important to note that VHDL is case-insensitive; and therefore `test' and `TEST' are the same. **The user is advised not to use identifiers differing only in case in the Verilog file as this may result in incorrect VHDL.**

verilog2vhdl has a list of reserved words which are not allowed as identifiers in input Verilog code. When they appear in the input Verilog code, **verilog2vhdl** creates legal VHDL names by modifying each reserved identifier to a VHDL extended identifier with the same name. For eg., VHDL reserved identifier `shared' would be modified to `\shared\'; all occurences of `shared' will be changed to `\shared\'. Reserved identifiers follow below:

```
abs                     false       nand
rol
access                  file                    new
ror
after                   function                next
select
alias                   generate                nor
severity
all                     generic                 not
signaland
architecture            guarded                 of
sla
```

```
array                   impure                  on
sll
assert                  in                      open
sra
attribute               inertial                others
srl
block                   inout                   out
subtype
body                    is                      package
then
buffer                  label                   port
to
bus                     library                 postponed
transport
component               linkage                 procedure
true
configuration           literal                 process
type
constant                loop                    pure
unaffected
disconnect              map                     range
units
downto                  mod                     record
until
elsif                                           register
use
entity                                          reject
variable
exit                                            rem
wait
                                                report
when
                                                return
with

xnor
                                                                        xor
```

Chapter3:
Using verilog2vhdl

Command Summary | Troubleshooting ||| Table of Contents

This chapter summarizes the options you can use to tailor **verilog2vhdl** to your specific needs, and describes steps to take if you encounter problems.

Command Summary

verilog2vhdl provides several options that allow you to tailor the operation of the tool. This section summarizes the commands you use to control these options.

```
Usage: v2v input_file [output_file] [options] [-Help] [-Usage] [-
Version]

where options can be a combination of

[-Replace] [-Silent] [-No_Package] [-No_Extract_comments]
[-Environment {Mentor|Synopsys|Generic}] [-Package {HDL files}] [-Log
{logfile_name}]
[-87|-93] [-No_Component_check] [-Component_check] [-SYNTH] [-
Map_Regs_to_Variables]
[-No_Synth] [-No_Zero_wait] [-Make_Defines_Constants] [-
Make_Parameters_Constants]
[-Reserved_Identifier_Prefix {prefix string}] [-
Reserved_Identifier_Suffix {suffix string}]
[-Preserve_Order] [-Verilog_PreProcessing] [-Architecture_Name] [-
No_header]

OPTIONS

-Replace (-r):
   Replace the existing output_file; default is to backup the
output_file
   to <output_file>.old

-Silent (-s):
   Supress printing out of messages indicating translator actions

-No_Package (-np):
   Valid with the [-Package] option; suppresses writing out of VHDL
packages
   of Verilog files supplied with [-Package] option; default is to
create VHDL
   package with filename <pkgfilename_without_extension>_pack.hdl

-Environment {Mentor|Synopsys|Generic} (-e {m|s|g}):
   Prints VHDL which is compliant with specified option, default is
Generic

-Package {HDL files} (-p {HDL files}):
   Loads the HDL files specified with this option into database

-Log {logfile_name} (-l {logfile name}):
   Logs translator messages into file <logfile_name>; default log file
is 'v2v.log'

-87:
   Produces VHDL compatible with 1076-1987 compliant simulator

-93:
   Produces VHDL compatible with 1076-1993 compliant simulator; this is
the default

-No_Extract_comments (-ne):
   Does not preserve comments in Verilog input

-No_Component_check (-nc):
```

```
   Does not look for module declarations for modules instantiated in
instantiations
   This is useful for translating designs which use large libraries

-Component_check (-c):
   Look for module declarations for modules instantiated in
instantiations
   This is the default, it can be used to override information in the
'v2v.ini' file

-SYNTH (-synth):
   Produces synthesizable VHDL

-Map_Regs_to_Variables (-mrv):
   Produces synthesizable VHDL with procedural assignments being mapped
to
   variable assignments

-No_Synth (-ns):
   Produces functionally equivalent simulatable VHDL; may not be
synthesizable
   can be used to override information in the 'v2v.ini' file

-No_Zero_wait (-nz):
   Does not print 'WAIT FOR 0 ns;' overridden by -synth option

-Make_Defines_Constants (-mdc):
   Makes all `defines encountered in the input file constants in the
Architecture;
   default is to create Generics in Entity

-Make_Parameters_Constants (-mpc):
   Makes all parameters encountered in the input file constants in the
Architecture;
   default is to create Generics in Entity

-Reserved_Identifier_Prefix <prefix string> (-rip <prefix string>):
   Prefixes specified string to a reserved verilog2vhdl identifier
   (See User's Manual, Chapter 3) for VHDL compliance

-Reserved_Identifier_Suffix <suffix string> (-ris <suffix string>):
   Suffixes specified string to a reserved verilog2vhdl identifier
   (See User's Manual, Chapter 3) for VHDL compliance

-Preserve_Order (-po):
   When printing VHDL, retain the order of concurrent constructs in the
Verilog input;
   default is to format the output VHDL

-Verilog_PreProcessing (-vpp):
   Preprocess the Verilog files before translatng themn so that the
Verilog compile directives
  can be elaborated andetherefore  supported

-Architecture_Name (-an):
   Set the name of the generated VHDL architecure.
   The default name is "VeriArch"
```

```
-No_header (-nh):
Do not print ASC header in the output file.
```

When you have problems...

If you experience problems using verilog2vhdl, please check the following:

- The input has the correct Verilog syntax. Errors will look like:

  ```
  // ERROR (file 'aha.v', line 2): missing '('
  // ERROR (file 'aha.v'): Syntax checks failed for file 'aha.v'
  // ERROR: Errors found in input file 'aha.v'
  ```

  If you don't know what the problem is, please reference the Verilog LRM. Make sure that there are no syntax errors in the input Verilog file.

- The construct causing the error is supported in the current version of verilog2vhdl.

- Read the document ‘Supported constructs’ and ‘known limitations’ provided with the tool. If you still have problems translating, then contact ASC support.

Chapter 4:
Compiling and simulating output VHDL

Order of Compilation | RTL Translation Example | Gates and Parametric Delay Example | '-Package' Switch Example ||| Table of Contents

The VHDL produced by **verilog2vhdl** needs to be run through a VHDL simulation and/or synthesis tool. This chapter describes the compilation order for the VHDL files followed by some examples illustrating the concept.

In order to facilitate translation, ASC has provided several packages along with the tool. Often times these packages are used in the translated file. User must compile these packages before compiling the output file.

If the output VHDL file does not use any of these packages then there is no need to compile any of the packages supplied with **verilog2vhdl**. You can directly proceed to compiling the VHDL output file.

In the ensuing discussion, **v2v_location** will be used to denote the location of the software on the host machine. **vhdl_sim** is used for the VHDL simulator.

verilog2vhdl comes with the following packages: **timing, buffers, numeric_std, v2v_types, utils** and **functions**. The **timing** package resides in the location *v2v_location/lib/timing.vhd ,* **buffers** package in *v2v_location/lib/buffers.vhd ,* **numeric_std** package in *v2v_location/lib/numeric_std.vhd,* **v2v_types** package in *v2v_location/lib/types.vhd*, **utils** package in *v2v_location/lib/utils.vhd* and the **functions** package in *v2v_location/lib/functions.vhd.* (If the `-87' switch is used, the related packages files are *v2v_location/lib/numeric_std_87.vhd* for the **numeric_std** package and *v2v_location/lib/functions_87.vhd* for the **functions** package.)

Order of Compilation for verilog2vhdl Packages

Chapter Start

verilog2vhdl uses tool-specific packages to perform the translation. In the output VHDL, these packages are grouped under a VHDL logical library by name **ASC**. The user needs to map the physical location of the compiled packages to this logical library by using VHDL compiler specific commands. Please refer to your VHDL simulator manual for information on library mapping. Additionally, for using IEEE STD 1076-1993 compliant VHDL, some VHDL compilers need extra switches. The compilation order for the VHDL packages is shown below. (Packages whose compile order is not important are grouped together. **vhdl_sim** is used as the shell invocation command for the VHDL simulator/compiler. Filenames of packages match exactly with the packages supplied with the tool.)

1. Package(s) **image_functions timing buffers**

   File(s) *image_functions.vhd timing.vhd buffers.vhd*

   command:

   % **vhdl_sim** image_functions.vhd
   % **vhdl_sim** timing.vhd
   % **vhdl_sim** buffers.vhd

2. Package(s) **numeric_std**

   File(s) *numeric_std.vhd (FOR 1993 VHDL)* **OR** *numeric_std_87.vhd (FOR 1987 VHDL)*

   command:

% **vhdl_sim** numeric_std.vhd
OR
% **vhdl_sim** numeric_std_87.vhd

3. Package(s) **v2v_types utils functions readmem**

   File(s) *types.vhd utils.vhd functions.vhd (FOR 1993 VHDL)* **OR***functions_87.vhd (FOR 1987 VHDL) readmem.vhd*

   command:

   % **vhdl_sim** types.vhd
   % **vhdl_sim** utils.vhd
   % **vhdl_sim** functions.vhd
   OR
   % **vhdl_sim** functions_87.vhd
   % **vhdl_sim** readmem.vhd

RTL Translation Example

Input File | verilog2vhdl Invocation and Transcript | Output File | VHDL Compilation ||| Chapter Start

Input file: *example.v*

Input File | verilog2vhdl Invocation and Transcript | Output File | VHDL Compilation ||| Section Start

```
////////////////////////////////////////////////////
// dir_cell component description
////////////////////////////////////////////////////

module dir_cell (so,shift,clock,si,idcode);
  output so;
  input shift,idcode,si,clock;

  reg cq;
  wire d,shift,idcode,si,clock;

  assign d = (shift & si) | (~shift & idcode);
  assign so = cq;

  always
      @(posedge clock) cq = d;

endmodule
```

verilog2vhdl Invocation and Transcript

Input File | verilog2vhdl Invocation and Transcript | Output File | VHDL Compilation ||| Section Start

```
% v2v example.v
```

```
// Copyright  (c) Alternative System Concepts Inc. 1992-2002, All
Rights Reserved.
//                        UNPUBLISHED, LICENSED SOFTWARE.
//             CONFIDENTIAL AND PROPRIETARY INFORMATION WHICH IS THE
//           PROPERTY OF ALTERNATIVE SYSTEM CONCEPTS OR ITS LICENSORS.
//
//
// Running..
// NOTE: Adding VHDL entity 'dir_cell' to output file
// NOTE: Adding VHDL architecture 'VeriArch' for entity 'dir_cell'
// NOTE: Adding port 'so' to entity 'dir_cell'
// NOTE: Adding port 'shift' to entity 'dir_cell'
// NOTE: Adding port 'clock' to entity 'dir_cell'
// NOTE: Adding port 'si' to entity 'dir_cell'
// NOTE: Adding port 'idcode' to entity 'dir_cell'
// NOTE: Adding Process in architecture 'VeriArch' belonging to entity
'dir_cell'
//

<<<<<< VHDL translation of Verilog file 'example.v' being written to
'example.hdl' >>>>>>
```

Output file: *example.hdl*

Input File | verilog2vhdl Invocation and Transcript | Output File | VHDL Compilation ||| Section Start

```
-------------------------------------------------------------------------
---------
--
-- File Type:    VHDL
-- Tool Version: verilog2vhdl  v1.0  Mon Feb  6 16:59:45 EST 1995
-- Date Created: Mon Feb 13 15:08:29 1995
--
-------------------------------------------------------------------------
--------

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY dir_cell IS
PORT (SIGNAL so : OUT std_logic;
      SIGNAL shift : IN std_logic;
      SIGNAL clock : IN std_logic;
      SIGNAL si : IN std_logic;
      SIGNAL idcode : IN std_logic);
END dir_cell;

ARCHITECTURE VeriArch OF dir_cell IS
SIGNAL cq : std_logic REGISTER ;
SIGNAL d : std_logic;
BEGIN
   d <= (shift and si) or (not shift and idcode);
   so <= cq;

PROCESS
BEGIN
   WAIT UNTIL posedge(clock);
   cq <= d;
```

```
END PROCESS;
END VeriArch;
```

VHDL Compilation

Input File | verilog2vhdl Invocation and Transcript | Output File | VHDL Compilation ||| Section Start

```
% vhdl_sim example.hdl
```

The above command will compile the VHDL code in file *example.hdl* in the current directory. If it is required to compile in a separate directory, check the simulator manual for options.

Gates and Parameteric Delay Example

Input File | verilog2vhdl Invocation and Transcript | Output File | VHDL Compilation ||| Chapter Start

Input File : *example.v*

Input File | verilog2vhdl Invocation and Transcript | Output File | VHDL Compilation ||| Chapter Start

```
/*********************************************************************

Verilog example illustrating gate instantiations

*********************************************************************/
`timescale 1ns/10ns

module gates(q,qb,d,clk,rst);
parameter delay_val=3;
output q,qb;
input d,clk,rst;

wire a,b,c,d; /*wire declarations*/
wire e,f,g,h;

nand #delay_val (qb,cf,clk,rst); /*nand gate instantiation*/
or #(1:2:3)  (qb,cf,clk,rst); /*nand gate instantiation*/
xor #10 (q,a,b,e,f,g); /*xor gate*/
xor #1 (q,a,b,e,f,g); /*xor gate*/
and #2 (b,d,e,f,g); /*and gate*/
buf #4 (e,f); /*buf*/
not #4 (d,g,h); /*not gate*/
notif0 firstnot (d,g,h); /*not gate*/
endmodule
```

verilog2vhdl Invocation and Transcript

Input File | verilog2vhdl Invocation and Transcript | Output File | VHDL Compilation ||| Chapter Start

```
% v2v example.v
// verilog2vhdl  v1.0  Mon Feb  6 16:59:45 EST 1995

// Copyright  (c) Alternative System Concepts Inc. 1992-1995, All
Rights Reserved.
//                        UNPUBLISHED, LICENSED SOFTWARE.
//            CONFIDENTIAL AND PROPRIETARY INFORMATION WHICH IS THE
```

```
//          PROPERTY OF ALTERNATIVE SYSTEM CONCEPTS OR ITS LICENSORS.
//
//
// Running..

// NOTE: Found compiler directive 'timescale'

// NOTE: Adding VHDL entity 'gates' to output
file
// NOTE: Adding VHDL architecture 'VeriArch' for entity
'gates'
// NOTE: Adding port 'q' to entity
'gates'
// NOTE: Adding port 'qb' to entity
'gates'
// NOTE: Adding port 'd' to entity
'gates'
// NOTE: Adding port 'clk' to entity
'gates'
// NOTE: Adding port 'rst' to entity
'gates'
// NOTE: Adding VHDL generic 'delay_val' of type NATURAL to entity
'gates'
// NOTE: Adding equivalent VHDL signal assignment for gate
instantiation 'firstnot'
//
      <<<<<< VHDL translation of Verilog file 'example.v' being written
to 'example.hdl' >>>>>>>
//
```

Output File : *example.hdl*

Input File | verilog2vhdl Invocation and Transcript | Output File | VHDL Compilation ||| Chapter Start

```
-------------------------------------------------------------------------
---------
--
-- File Type:    VHDL
-- Tool Version: verilog2vhdl  v1.0  Mon Feb  6 16:59:45 EST 1995
-- Date Created: Mon Feb 13 15:11:35 1995
--
-------------------------------------------------------------------------
--------

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY gates IS
GENERIC (CONSTANT delay_val : natural := 3);
PORT (SIGNAL q : OUT std_logic;
      SIGNAL qb : OUT std_logic;
      SIGNAL d : IN std_logic;
      SIGNAL clk : IN std_logic;
      SIGNAL rst : IN std_logic);
END gates;

LIBRARY ASC;
ARCHITECTURE VeriArch OF gates IS
```

```
SIGNAL e : std_logic;
SIGNAL f : std_logic;
SIGNAL g : std_logic;
SIGNAL h : std_logic;
COMPONENT notif0


GENERIC (CONSTANT bufdelay : time);
PORT (SIGNAL output : OUT std_logic;
      SIGNAL input : IN std_logic;
      SIGNAL enable : IN std_logic);
END COMPONENT;
USE ASC.buffers.all;
FOR ALL : notif0 USE ENTITY Verilog.notif0;

USE ASC.timing.all;
USE ASC.functions.all;
SIGNAL cf : std_logic;
SIGNAL a : std_logic;
SIGNAL b : std_logic;
SIGNAL c : std_logic;
BEGIN
   qb <= ((cf nand clk) nand rst) AFTER parameter_delay(delay_val, ns);
   qb <= ((cf or clk) or rst) AFTER delay(1, 2, 3, ns);
   q <= ((((a xor b) xor e) xor f) xor g) AFTER 10ns;
   q <= ((((a xor b) xor e) xor f) xor g) AFTER 1ns;
   b <= (((d and e) and f) and g) AFTER 2ns;
   e <= f AFTER 4ns;
   g <= not h AFTER 4ns;
   d <= not h AFTER 4ns;
   firstnot : Verilog.buffers.notif0 PORT MAP (d, g, h);
END VeriArch;
```

VHDL Compilation

Input File | verilog2vhdl Invocation and Transcript | Output File | VHDL Compilation ||| Chapter Start

```
% vhdl_sim buffers.vhd

% vhdl_sim timing.vhd


% vhdl_sim numeric_std.vhd


% vhdl_sim functions.vhd


% vhdl_sim example.hdl <map logical library `Verilog' to the location
where the compilation results of `buffers', `timing' , `numeric_std'
and `functions' package(s) lie>
```

Since these packages are used extensively in **verilog2vhdl,** it is suggested that the user pre-compile these packages. This will allow skipping the first few steps in the compilation for every run of **verilog2vhdl** that uses these packages.

'-Package' switch usage Example

Input File | Input File 2 | verilog2vhdl Invocation and Transcript | Output File | Output File 2 | VHDL Compilation ||| Chapter Start

Input File: *adders.v*

Input File | Input File 2 | verilog2vhdl Invocation and Transcript | Output File | Output File 2 | VHDL Compilation ||| Section Start

```
// Base F-T 2-bit adder design with internal scan

module adders(a1, b1, a2, b2, s, test_clk, scan_en, scan_in0, scan_out,
            scan_in1);

input [1:0] a1, b1, a2, b2;
input scan_in0, scan_in1;
input test_clk, scan_en;
output s;
output [1:0] scan_out;
parameter wire_width = 2;
parameter scan_out_width = 1;

reg s;
reg [scan_out_width:0] scan_out;
wire [wire_width:0] s1, s2;

    half_adder       add1 (.s(s1[0]), .co(add1co), .a(a1[0]),
.b(b1[0])),
                add3 (.s(s2[0]), .co(add3co), .a(a2[0]), .b(b2[0]));
    aha       add2 (.s(s1[1]), .co(s1[2]), .a(a1[1]), .b(b1[1]),
.c(add1co)),
                add4 (.s(s2[1]), .co(s2[2]), .a(a2[1]), .b(b2[1]),
.c(add3co));

    sc_mux       cell1 (.D(s1[2]), .Q(cell1out), .clk(test_clk),
.S_in(scan_in0),
                  .S_en(scan_en)),
                cell2 (.D(s1[1]), .Q(cell2out), .clk(test_clk),
.S_in(cell1out),
                  .S_en(scan_en)),
                cell3 (.D(s1[0]), .Q(cell3out), .clk(test_clk),
.S_in(cell2out),
                  .S_en(scan_en)),
                cell4 (.D(s2[2]), .Q(cell4out), .clk(test_clk),
.S_in(scan_in1),
                  .S_en(scan_en)),
                cell5 (.D(s2[1]), .Q(cell5out), .clk(test_clk),
.S_in(cell4out),
                  .S_en(scan_en)),
                cell6 (.D(s2[0]), .Q(cell6out), .clk(test_clk),
.S_in(cell5out),
                  .S_en(scan_en));

     always @(cell3out)
          scan_out[0] = cell3out;

     always @cell6out
```

```
          scan_out[1] = cell6out;

     // voter
     always @(cell1out or cell2out or cell3out or cell4out or cell5out
or cell6out)
      begin
          if ((cell1out == cell4out) & (cell2out == cell5out) &
            (cell3out == cell6out))
               s = 1'b1;
          else
               s = 1'b0;
      end
endmodule
```

Input files (2): *files.v*

Input File | Input File 2 | verilog2vhdl Invocation and Transcript | Output File | Output File 2 | VHDL Compilation ||| Section Start

```
// Verilog file for bit-wide adders

module aha(a,b,c,s,co);
input a,b,c;
output s,co;

parameter


      s_delay_min = 3, s_delay_typ = 4, s_delay_max = 5;

xor (a,b,c);
assign #(s_delay_min : s_delay_typ : s_delay_max) s = a;
assign #4 co = (a & b) ^ (a & c) ^ (b & c);

endmodule

module half_adder(a,b,s,co);
input a,b;
output s,co;

parameter delay = 2;

assign #delay s = a ^ b;
and(a,b);
assign #delay co = a;

endmodule

// Verilog model of a muxed scan cell

module sc_mux(D, Q, clk, S_in, S_en);
input D, clk, S_in, S_en;
output Q;
reg Q;
wire dff_d;

mux2 mux (S_in, D, S_en, dff_d);
```

```
always @(posedge clk)
      #1 Q = dff_d;

endmodule

module mux2(a,b,en,y);
input a,b,en;
output y;
reg y;

always @(a or b or en)
begin
      if (en == 1)
            y = a;
      else if (en == 0)
            y = b;
      else
            y = 1'bz;
end
endmodule
```

verilog2vhdl Invocation and Transcript

Input File | Input File 2 | verilog2vhdl Invocation and Transcript | Output File | Output File 2 | VHDL Compilation ||| Section Start

```
% v2v adders.v -p files.v
// verilog2vhdl  v1.0  Mon Feb  6 16:59:45 EST 1995

// Copyright  (c) Alternative System Concepts Inc. 1992-1995, All
Rights Reserved.
//                        UNPUBLISHED, LICENSED SOFTWARE.
//            CONFIDENTIAL AND PROPRIETARY INFORMATION WHICH IS THE
//          PROPERTY OF ALTERNATIVE SYSTEM CONCEPTS OR ITS LICENSORS.
//
//
// Running..
// NOTE: Adding VHDL entity 'adders' to output
file
// NOTE: Adding VHDL architecture 'VeriArch' for entity 'adders'
// NOTE: Adding port 'a1' to entity 'adders'
// NOTE: Adding port 'b1' to entity 'adders'
// NOTE: Adding port 'a2' to entity 'adders'
// NOTE: Adding port 'b2' to entity 'adders'
// NOTE: Adding port 's' to entity 'adders'
// NOTE: Adding port 'test_clk' to entity 'adders'
// NOTE: Adding port 'scan_en' to entity 'adders'
// NOTE: Adding port 'scan_in0' to entity 'adders'
// NOTE: Adding port 'scan_out' to entity 'adders'
// NOTE: Adding port 'scan_in1' to entity 'adders'
// NOTE: Adding VHDL generic 'wire_width' of type NATURAL to entity
'adders'
// NOTE: Adding VHDL generic 'scan_out_width' of type NATURAL to entity
'adders'
// NOTE: Adding module instantiation 'add1' of module 'half_adder' in
architecture 'VeriArch'
// NOTE: Adding module instantiation 'add3' of module 'half_adder' in
architecture 'VeriArch'
```

```
// NOTE: Adding module instantiation 'add2' of module 'aha' in
architecture 'VeriArch'
// NOTE: Adding module instantiation 'add4' of module 'aha' in
architecture 'VeriArch'
// NOTE: Adding module instantiation 'cell1' of module 'sc_mux' in
architecture 'VeriArch'
// NOTE: Adding module instantiation 'cell2' of module 'sc_mux' in
architecture 'VeriArch'
// NOTE: Adding module instantiation 'cell3' of module 'sc_mux' in
architecture 'VeriArch'
// NOTE: Adding module instantiation 'cell4' of module 'sc_mux' in
architecture 'VeriArch'
// NOTE: Adding module instantiation 'cell5' of module 'sc_mux' in
architecture 'VeriArch'
// NOTE: Adding module instantiation 'cell6' of module 'sc_mux' in
architecture 'VeriArch'
// NOTE: Adding Process in architecture 'VeriArch' belonging to entity
'adders'
// NOTE: Adding Process in architecture 'VeriArch' belonging to entity
'adders'
// NOTE: Adding Process in architecture 'VeriArch' belonging to entity
'adders'
//
<<<<<< VHDL translation of Verilog file 'adders.v' being written to
'adders.hdl' >>>>>>>
//
//
<<<<<< VHDL package of Verilog file 'files.v' being written to
'files_pack.hdl' >>>>>>>>
```

Ouput file: *adders.hdl*

Input File | Input File 2 | verilog2vhdl Invocation and Transcript | Output File | Output File 2 | VHDL Compilation ||| Section Start

```
-------------------------------------------------------------------------
---------
--
-- File Type:    VHDL
-- Tool Version: verilog2vhdl  v1.0  Mon Feb  6 16:59:45 EST 1995
-- Date Created: Mon Feb 13 15:15:31 1995
--
-------------------------------------------------------------------------
--------

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY adders IS
GENERIC (CONSTANT wire_width : natural := 2;
         CONSTANT scan_out_width : natural := 1);
PORT (SIGNAL a1 : IN std_logic_vector(1 DOWNTO 0);
      SIGNAL b1 : IN std_logic_vector(1 DOWNTO 0);
      SIGNAL a2 : IN std_logic_vector(1 DOWNTO 0);
      SIGNAL b2 : IN std_logic_vector(1 DOWNTO 0);
      SIGNAL s : OUT std_logic;
      SIGNAL test_clk : IN std_logic;
      SIGNAL scan_en : IN std_logic;
      SIGNAL scan_in0 : IN std_logic;
```

```
      SIGNAL scan_out : OUT std_logic_vector(1 DOWNTO 0);
      SIGNAL scan_in1 : IN std_logic);
END adders;

LIBRARY user_defined;
ARCHITECTURE VeriArch OF adders IS

FOR ALL : half_adder USE ENTITY user_defined.half_adder;
FOR ALL : sc_mux USE ENTITY user_defined.sc_mux;

SIGNAL s1 : std_logic_vector(wire_width DOWNTO 0);
SIGNAL cell3out : std_logic;
SIGNAL cell2out : std_logic;
SIGNAL cell1out : std_logic;
SIGNAL add1co : std_logic;

FOR ALL : aha USE ENTITY user_defined.aha;

USE user_defined.files_pack.all;
SIGNAL s2 : std_logic_vector(wire_width DOWNTO 0);
SIGNAL cell6out : std_logic;
SIGNAL cell5out : std_logic;
SIGNAL cell4out : std_logic;
SIGNAL add3co : std_logic;
BEGIN
   add1 : user_defined.files_pack.half_adder PORT MAP
                (s => s1(0),
                     co => add1co,
                     a => a1(0),
                     b => b1(0)) ;
   add3 : user_defined.files_pack.half_adder PORT MAP
                (s => s2(0),
                     co => add3co,
                     a => a2(0),
                     b => b2(0)) ;
   add2 : user_defined.files_pack.aha PORT MAP
                (s => s1(1),
                     co => s1(2),
                     a => a1(1),
                     b => b1(1),
                     c => add1co) ;
   add4 : user_defined.files_pack.aha PORT MAP
                (s => s2(1),
                     co => s2(2),
                     a => a2(1),
                     b => b2(1),
                     c => add3co) ;
   cell1 : user_defined.files_pack.sc_mux PORT MAP
                (D => s1(2),
                     Q => cell1out,
                     clk => test_clk,
                     S_in => scan_in0,
                     S_en => scan_en) ;
   cell2 : user_defined.files_pack.sc_mux PORT MAP
                (D => s1(1),
                     Q => cell2out,
                     clk => test_clk,
```

```
                    S_in => cell1out,
                    S_en => scan_en)
;
   cell3 : user_defined.files_pack.sc_mux PORT MAP
                (D => s1(0),
                    Q => cell3out,
                    clk => test_clk,
                    S_in => cell2out,
                    S_en => scan_en) ;
   cell4 : user_defined.files_pack.sc_mux PORT MAP
                (D => s2(2),
                    Q => cell4out,
                    clk => test_clk,
                    S_in => scan_in1,
                    S_en => scan_en) ;
   cell5 : user_defined.files_pack.sc_mux PORT MAP
                (D => s2(1),
                    Q => cell5out,
                    clk => test_clk,
                    S_in => cell4out,
                    S_en => scan_en) ;
   cell6 : user_defined.files_pack.sc_mux PORT MAP
                (D => s2(0),
                    Q => cell6out,
                    clk => test_clk,
                    S_in => cell5out,
                    S_en => scan_en) ;

PROCESS
BEGIN
   WAIT ON cell3out;
   scan_out(0) <= cell3out;
END PROCESS;

PROCESS
BEGIN
   WAIT ON cell6out;
   scan_out(1) <= cell6out;
END PROCESS;

PROCESS
BEGIN
   WAIT ON cell1out, cell2out, cell3out, cell4out, cell5out, cell6out;
   IF ((cell1out = cell4out) and (cell2out = cell5out) and
      (cell3out = cell6out)) THEN
      s <= '1';
   ELSE
      s <= '0';
   END IF;
END PROCESS;
END VeriArch;
```

Output File 2: *files_pack.hdl*

Input File | Input File 2 | verilog2vhdl Invocation and Transcript | Output File | Output File 2 | VHDL Compilation ||| Section Start

```
--------------------------------------------------------------------------
---------
--
-- File Type:    VHDL
-- Tool Version: verilog2vhdl  v1.0  Mon Feb  6 16:59:45 EST 1995
-- Date Created: Mon Feb 13 15:15:31 1995
--
--------------------------------------------------------------------------
--------

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY mux2 IS
PORT (SIGNAL a : IN std_logic;
      SIGNAL b : IN std_logic;
      SIGNAL en : IN std_logic;
      SIGNAL y : OUT std_logic);
END mux2;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY aha IS
GENERIC (CONSTANT s_delay_min : natural := 3;
         CONSTANT s_delay_typ : natural := 4;
         CONSTANT s_delay_max : natural := 5);
PORT (SIGNAL a : IN std_logic;
      SIGNAL b : IN std_logic;
      SIGNAL c : IN std_logic;
      SIGNAL s : OUT std_logic;
      SIGNAL co : OUT std_logic);
END aha;

LIBRARY ASC;
ARCHITECTURE VeriArch OF aha IS
USE ASC.timing.all;
BEGIN
   a <= (b xor c);
   s <= a AFTER delay(s_delay_min, s_delay_typ, s_delay_max, NS);
   co <= (a and b) xor (a and c) xor (b and c) AFTER 4NS;
END VeriArch;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY half_adder IS
GENERIC (CONSTANT delay : natural := 2);
PORT (SIGNAL a : IN std_logic;
      SIGNAL b : IN std_logic;
      SIGNAL s : OUT std_logic;
      SIGNAL co : OUT std_logic);
END half_adder;

LIBRARY ASC;
ARCHITECTURE VeriArch OF half_adder IS
USE ASC.timing.all;
BEGIN
   s <= a xor b AFTER parameter_delay(delay, NS);
   a <= b;
```

```
   co <= a AFTER parameter_delay(delay, NS);
END VeriArch;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY sc_mux IS
PORT (SIGNAL D : IN std_logic;
      SIGNAL Q : OUT std_logic;
      SIGNAL clk : IN std_logic;
      SIGNAL S_in : IN std_logic;
      SIGNAL S_en : IN std_logic);
END sc_mux;

ARCHITECTURE VeriArch OF sc_mux IS
SIGNAL dff_d : std_logic;
COMPONENT mux2
PORT (SIGNAL a : IN std_logic;
      SIGNAL b : IN std_logic;
      SIGNAL en : IN std_logic;
      SIGNAL y : OUT std_logic);
END COMPONENT;
FOR ALL : mux2 USE ENTITY work.mux2;

BEGIN

PROCESS
BEGIN
   WAIT UNTIL posedge(clk);
   WAIT FOR 1NS;
   Q <= dff_d;
END PROCESS;
   mux : mux2 PORT MAP (S_in, D, S_en, dff_d) ;
END VeriArch;

ARCHITECTURE VeriArch OF mux2 IS
BEGIN

PROCESS
BEGIN
   WAIT ON a, b, en;
   IF en = '1' THEN
      y <= a;
   ELSE
      IF en = '0' THEN
         y <= b;
      ELSE
         y <= 'Z';
      END IF;
   END IF;
END PROCESS;
END VeriArch;

LIBRARY ieee;
USE ieee.std_logic_1164.all;
PACKAGE files_pack IS
COMPONENT mux2
PORT (SIGNAL a : IN std_logic;
```

```
      SIGNAL b : IN std_logic;
      SIGNAL en : IN std_logic;
      SIGNAL y : OUT std_logic);
END COMPONENT;
COMPONENT aha
GENERIC (CONSTANT s_delay_min : natural := 3;
         CONSTANT s_delay_typ : natural := 4;
         CONSTANT s_delay_max : natural := 5);
PORT (SIGNAL a : IN std_logic;
      SIGNAL b : IN std_logic;
      SIGNAL c : IN std_logic;
      SIGNAL s : OUT std_logic;
      SIGNAL co : OUT std_logic);
END COMPONENT;
COMPONENT half_adder
GENERIC (CONSTANT delay : natural := 2);
PORT (SIGNAL a : IN std_logic;
      SIGNAL b : IN std_logic;
      SIGNAL s : OUT std_logic;
      SIGNAL co : OUT std_logic);
END COMPONENT;
COMPONENT sc_mux
PORT (SIGNAL D : IN std_logic;
      SIGNAL Q : OUT std_logic;
      SIGNAL clk : IN std_logic;
      SIGNAL S_in : IN std_logic;
      SIGNAL S_en : IN std_logic);
END COMPONENT;
COMPONENT mux2
PORT (SIGNAL a : IN std_logic;
      SIGNAL b : IN std_logic;
      SIGNAL en : IN std_logic;
      SIGNAL y : OUT std_logic);
END COMPONENT;
END;
```

VHDL Compilation

Input File | Input File 2 | verilog2vhdl Invocation and Transcript | Output File | Output File 2 | VHDL Compilation ||| Section Start

verilog2vhdl writes out VHDL packages for the Verilog files read in by the `-Package' switch. The name of the VHDL package is `<pkg_filename_without_extension>_pack' and the VHDL filename is <pkg_filename_without_extension>_pack.hdl. For e.g, if a Verilog file of the name files.v is read into the verilog2vhdl database, the subsequent VHDL package `files_pack' would be placed in file files_pack.hdl.

```
% vhdl_sim timing.vhd

% vhdl_sim files_pack.hdl

% vhdl_sim adders.hdl <map Logical library `user_defined' to the VHDL
compilation results of `files' package>
```

In the case when there is more than one logical library in the final VHDL, each logical library needs to be mapped the the physical location of VHDL compilation results.