We can use VHDL not only as a system design language, but also as a language to generate a simulation in ModelSim. A VHDL file that creates a simulation for another VHDL file is called a **testbench**.

For example, suppose we create a VHDL design entity for the equivalent of a 74138 TTL decoder. This decoder has three active-HIGH decode inputs and eight active-LOW outputs. It also has three enable inputs, all of which must be active to enable the decoder outputs. $G1$ is active-HIGH; $\overline{G2A}$ and $\overline{G2B}$ are active-LOW.

We will write some VHDL code to describe the decoder, then we will write a VHDL testbench to generate a simulation of the decoder.

VHDL Decoder:

```
-- dcd138.vhd
-- VHDL equivalent of a 74138 3-to-8 decoder
-- Outputs are active-LOW
-- Enable if: G1 and (not nG2A) and (not nG2B)

library ieee;
use ieee.std_logic_1164.all;

entity dcd138 is
   port(
      d     : in  std_logic_vector(2 downto 0);
      g1    : in  std_logic;
      ng2a  : in  std_logic;
      ng2b  : in  std_logic;
      y     : out std_logic_vector(0 to 7)
      );
end dcd138;

architecture decode of dcd138 is
   signal inputs : std_logic_vector(5 downto 0);
begin
   inputs <= g1 & ng2a & ng2b & d;

   with inputs select
      y <=  "01111111" when "100000",
            "10111111" when "100001",
            "11011111" when "100010",
            "11101111" when "100011",
            "11110111" when "100100",
            "11111011" when "100101",
            "11111101" when "100110",
            "11111110" when "100111",
            "11111111" when others;
end decode;
```

Testbench Structure

Figure 1 shows how a testbench is structurally connected to a VHDL design entity. The 74138-equivalent decoder (**dcd138.vhd**) is referred to as the Unit Under Test (UUT). The UUT is instantiated as a component in the testbench file (**dcd138_tb.vhd**). The testbench file itself has no input or output ports that connect to physical hardware in the outside world; the only connections the testbench has are an internal set of signals that connect to the ports of the UUT. Therefore its entity declaration contains no ports, only the testbench file name.

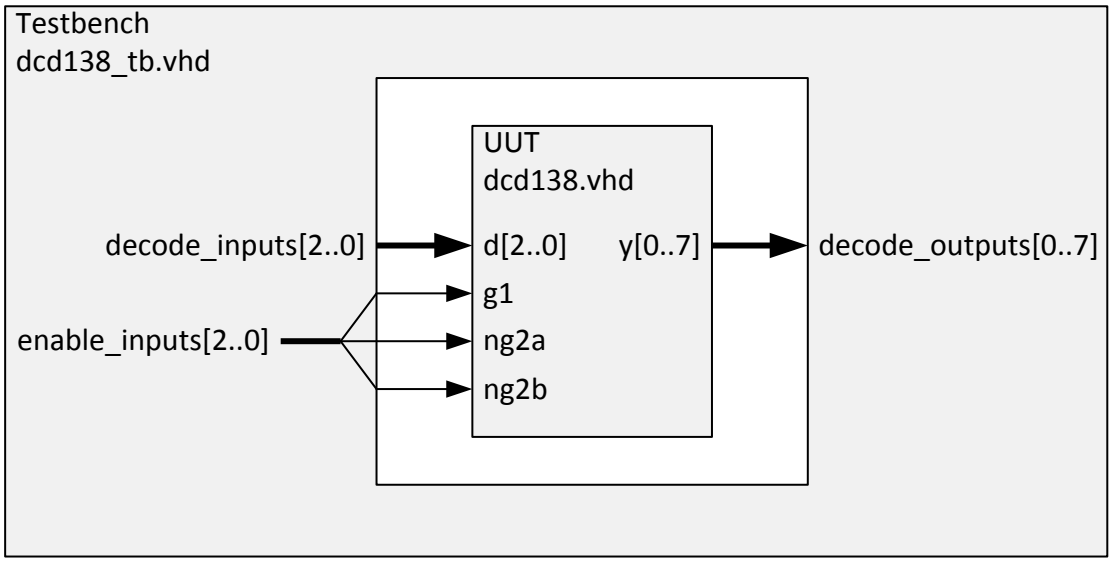

Figure 1 Structure of a VHDL Testbench for a 74138 Decoder

Table 1 summarizes the objects required to connect the testbench to the unit under test.

Table 1 Summary of Connections Between Testbench and Unit Under Test

	Unit Under Test	**Testbench**
Entity Name	dcd138	dcd138_tb
Ports	d(2 downto 0) g1, ng2a, ng2b y(0 to 7)	None (Entity contains entity name only; otherwise it is empty)
Signals	None shown in Figure 1. May be part of the decoder design.	decode_inputs(2 downto 0) enable_inputs(2 downto 0) decode_outputs(0 to 7)
Components	None	dcd138

Examine the VHDL code for the testbench on page 4. We will refer to the given line numbers to explain the functionality of the different parts of the testbench.

Line 10: In addition to the **std_logic_1164** package, we also use the **std_logic_arith** package, which contains conversion functions that allow us to convert between **std_logic** and **integer** types.

Lines 12 – 14: Declare the testbench entity, using testbench entity name only – no ports. A useful convention is to name the testbench the same as the unit under test, followed by _tb in the entity name.

Lines 17 – 25: Declare the unit under test (UUT) as a component in the testbench file. The UUT entity declaration and modified to make the component declaration.

Lines 27 – 29: Declare signals to connect from the testbench to the ports of the unit under test.

Lines 32 – 36: Instantiate the unit under test by mapping the UUT component ports to the testbench signals declared in lines 27 – 29. The result is the connection of the UUT to the testbench shown in Figure 1.

Lines 39 – 60: Generate the simulation signals by specifying changes on the testbench signals. Details are given below.

The simulation signals are generated by a VHDL **process**. Statements inside a process are **sequential**, that is, run in the order they are written, like a program in C. Timing in the process is specified by **wait** or **wait for** statements.

To test the operation of the decoder, it is logical to generate a binary sequence on the decode inputs of the UUT, both when the enable inputs are all active and also for at least several cases where they are not. The process that generates the waveforms, the so-called **stimulus process**, consists of an outer and an inner **for loop**, as follows:

```
for i in 1 to 4 loop
   -- set enable inputs by a case statement
   for j in 0 to 7 loop
      --set decode inputs by an assignment statement
   end loop;
end loop;
```

We are not testing all combinations of enable input, so it is logical to use a case statement to select the cases we want. In VHDL, the syntax of a **case** statement is:

```
case expression is
   when value_1_of_expression =>
      statement
      statement
   when value_2_of_expression =>
      statement
      statement
   when others =>
      statement
      statement
end case;
```

The decode input values must be of **std_logic** types, but the inner for loop generates integers, we can use a conversion function (on line 55) to create the required 3-bit **std_logic_vector** values.

```
1  -- dcd138_tb.vhd
2  -- test bench for dcd138.vhd
3  -- simulation criteria:
4  --    when enable inputs are g1 ng2a ng2b = "100",
5  --    subscript of active decoder output is equivalent to
6  --    binary value of decoder inputs
7
8  library ieee;
9  use ieee.std_logic_1164.all;
10 use ieee.std_logic_arith.all;
11
12 entity dcd138_tb is
13    -- test bench entity has no external ports
14 end dcd138_tb;
15
16 architecture test_bench of dcd138_tb is
17    component dcd138
18       port(
19          d  : in     std_logic_vector(2 downto 0);
20          g1 : in  std_logic;
21          ng2a  : in     std_logic;
22          ng2b  : in     std_logic;
23          y  : out std_logic_vector(0 to 7)
24          );
25    end component;
26
27    signal enable_inputs    : std_logic_vector(2 downto 0);
28    signal decode_inputs    : std_logic_vector(2 downto 0);
29    signal decode_outputs   : std_logic_vector(0 to 7);
30 begin
31 -- instantiate decoder with test signals
32 decoder: dcd138 port map(  d     => decode_inputs,
33                            g1    => enable_inputs(2),
34                            ng2a  => enable_inputs(1),
35                            ng2b  => enable_inputs(0),
36                            y     => decode_outputs);
37
38    -- stimulus process
39    stim_proc: process
40    begin
41       decode_inputs <= "000"; -- initialize inputs
42       for i in 1 to 4 loop
43          case(i) is
44             when 1 =>
45                enable_inputs <= "100";
46             when 2 =>
47                enable_inputs <= "000";
48             when 3 =>
49                enable_inputs <= "010";
50             when 4 =>
51                enable_inputs <= "001";
52          end case;
53          wait for 20 ns;
54          for j in 0 to 7 loop
55             decode_inputs <= CONV_STD_LOGIC_VECTOR(j,3);
56             wait for 100 ns;
57          end loop;
58       end loop;
59       wait;    -- stall here
60    end process;
61 end test_bench;
```