Chapter 2 VHDL for Engineers Solutions

revised June 28, 2008 8:42 pm

2.1 A design unit is a VHDL construct that can be separately compiled and stored in a design library. Design units provide modularity for design management of complex systems. A design unit consists of a context clause followed by a library unit. Compilation of a design unit defines the corresponding library unit, which is stored in a design library.

2.2 The five kinds of library units are: entity declaration, architecture body, package declaration, package body, and configuration declaration.

2.3 Separating a design entity into an entity declaration and an architecture body makes it easier to experiment with alternative implementations. Also, in a team design effort, this separation allows the entity declaration to be written by one person and the architecture body by another.

2.4 An entity declaration gives a design entity its name and defines its interface to the environment in which it is used. It names the entity's inputs and outputs and defines the type of information they carry. It provides all of the information needed to physically connect the design entity to the outside world or to connect it as a component in a larger system.

2.5 The symbol ::= means that the text to the left of the symbol can be replaced by the text to the right of the symbol in a production. The symbol | separates alternative items, unless it occurs immediately after an opening brace, in which case it stands for itself. The pair of symbols [ ] enclose optional items. The pair of symbols { } enclose repeated items. The enclosed items may be repeated zero or more times.

2.6 A synthesis tool is said to accept a VHDL construct if it allows the construct to be a legal input. In contrast, a synthesis tool is said to interpret a construct if it synthesizes hardware that represents the construct.

2.7 In terms of an IEEE Std 1076.6 compliant synthesizer, a VHDL construct is categorized as supported if the synthesizer interprets the construct (maps the construct to an equivalent hardware representation). A construct is categorized as ignored if the synthesizer ignores the construct and produces a warning. When the synthesizer encounters such a construct, the synthesis does not fail. However, synthesis results may not match simulation results. A construct is not supported if the synthesizer does not expect to encounter the construct and may fail. However, failure is not
required by the standard and a particular synthesizer may treat such a construct as ignored. Parts of a construct that are ignored are underlined in syntax definitions. Parts of a construct that are not supported are struck through in syntax definitions.

2.8 The name `result_` is invalid as an identifier in VHDL because it ends with an underscore. The name `4_to_1_mux` is invalid as an identifier because it does not start with a letter of the alphabet. The name `port` is invalid as an identifier because it is a keyword.

2.9 Mode linkage is not synthesizable. The modes and their direction of transfer are:

mode	direction(s) of transfer
in	in
out	out
inout	in and out
buffer	in and out
linkage	in and out

2.10 A source is a contributor to the value of a signal. In VHDL there are two kinds of sources, one is a port of mode out, inout, or buffer and the other is a driver. A driver is created by a signal assignment statement.

2.11 A signal assignment statement can write ports of mode out, inout, and buffer. A signal assignment statement can read ports of mode in, inout, and buffer.

2.12 Statement (a) is sensitive to signals `a`, `b`, and `c`, which are the signals that are read. Signal `f` is written. Statement (b) is sensitive to signals `w`, `x`, and `y`, which are the signals that are read. Signal `g` is written. Statement (c) is sensitive to signals `x`, and `y`, which are the signals that are read. Signal `eq` is written. Statement (d) is sensitive to signals `x_bar`, `y_bar`, `x`, and `y`, which are the signals that are read. Signal `eq` is written. Statement (e) is sensitive to signals `armed`, `d1`, `d2`,and `d3`, which are the signals that are read. Signal `alarm` is written.

2.13 Signals `s1` and `s2` are local signals, since they are declared in the declarative part of the architecture body. That leaves `a`, `b`, `c`, and `f` as ports. Ports `a`, `b`, and `c` are read, therefore, they are input ports. Port `f` is written, therefore, it is an output port.

```
entity and_or is
   port(
      a, b, c : in std_logic;
      f : out std_logic
      );
end and_or;
```

2.14 For a mode inout port driving an external signal that is also driven by other ports, the value written to the port is the value driven by the port. The value read from the port is determined from the resolution of the value driven by the port and the values driven by the other ports connected to the common external signal.

For a port with mode buffer that is driving an external signal that is also driven by other ports, the value that is written to the port is the value driven by the port. However, unlike mode inout, with mode buffer the value obtained when the port is read is the same as the value driven by the port. This value is not necessarily the same as the value of the external signal connected to the buffer port. The value of the external signal is determined from the resolution of the value driven by the port and the values driven by other ports connected to the external signal. A buffer port is useful when a signal value that is being output from an entity needs to be read within the entity's architecture body. This is a feedback situation that is useful in sequential systems.

2.15 An architecture body defines either the behavior or structure of a design entity. That is, how it accomplishes its function or how it is constructed. As such, it provides the internal view of a design entity.

2.16 An architecture body is divided into a declarative part and a statement part. The declarative part starts immediately following keyword `is` and ends at keyword `begin`. The declarative part is used to declare signals and other items used in the architecture. The statement part of an architecture body starts immediately following keyword `begin` and ends at keyword `end`. The statement part consists of one or more concurrent statements.

2.17 The three pure coding styles are: dataflow, behavioral, and structural. The distinction between these styles is based on the type of concurrent statements used. A dataflow architecture uses only concurrent signal assignment statements. A behavioral architecture uses only process statements. A structural architecture uses only component instantiation statements.

2.18
(a) and (b)
```
library ieee;
use ieee.std_logic_1164.all;

entity mux is
 port (a, b, s : in std_logic;
 c : out std_logic);
end;

architecture dataflow of mux is
begin

 c <= (not s and a) or (s and b);

end dataflow;
```

(c) The context clause is required in addition to the entity declaration and architecture body. Since type std_logic is defined in the package STD_LOGIC_1164.

2.19
(a)

$$x = \bar{a}\bar{b}c + \bar{a}b\bar{c} + a\bar{b}\bar{c} + abc$$
$$y = \bar{a}\bar{b}\bar{c} + a\bar{b}c + ab\bar{c}$$

(b)
```
library ieee;
use ieee.std_logic_1164.all;

entity comb_fcn is
   port (
      a, b, c : in std_logic;
      x, y : out std_logic
      );
end comb_fcn;

architecture comb_fcn of comb_fcn is
begin

   x <= (not a and not b and c) or (not a and b and not c) or
   (a and not b and not c) or (a and b and c);

   y <= (not a and not b and not c) or (a and not b and c) or
   (a and b and not c);

end comb_fcn;
```

2.20

```
library ieee;
use ieee.std_logic_1164.all;

entity truthtbl is
   port (a, b, c : in std_logic;
      x, y : out std_logic);
end;

architecture dataflow of truthtbl is
begin
   x <= (not a and not b and c)
   or (a and not b and not c);

   y <= (not a and not b and not c)
   or (not a and b and not c)
   or (a and b and c);

end dataflow;
```

2.21

```
library ieee;
use ieee.std_logic_1164.all;

entity triple_3 is
   port (
      a1, a2, a3, b1, b2, b3, c1, c2, c3: in std_logic;
      y1, y2, y3: out std_logic
      );
end triple_3;

architecture triple_3 of triple_3 is
begin

   y1 <= not (a1 and b1 and c1);
   y2 <= not (a2 and b2 and c2);
   y3 <= not (a3 and b3 and c3);
```

```
end triple_3;
```

2.22 In VHDL, a component is actually a placeholder for a design entity. A structural design that uses components simply specifies the interconnection of the components. Ultimately, the actual design entity that each component represents must be established.

2.23 The establishment of the actual design entity that each component represents is referred to as binding each component to a design entity. This binding can be specified explicitly or can occur by default. VHDL's default binding rules implicitly bind a component to a design entity having the same name and same port interface. Port interfaces are the same if the ports have identical names and types. If, in the working library, there is more than one architecture body associated with an entity declaration, then default binding causes the compiler to use the one most recently compiled.

2.24 Note: this solution uses positional association in the port maps.

```
library ieee;
use ieee.std_logic_1164.all;

entity top_level is
   port (
      a, b, c: in std_logic;
      y: out std_logic
      );
end top_level;

architecture top_level of top_level is
   signal s1, s2, s3 : std_logic;
begin

   u1: and_2 port map (a, b, s1);
   u2: and_2 port map (s1, c, s2);
   u3: not_1 port map (c, s3);
   u4: or_2 port map (s2, s3, y);

end top_level;
```

2.25

```
library ieee;
use ieee.std_logic_1164.all;

entity nand_2 is
   port (i1, i2 : in std_logic;
      o : out std_logic);
end nand_2;

architecture dataflow of nand_2 is
begin
   o <= i1 nand i2;
end dataflow;
```

```
library ieee;
use ieee.std_logic_1164.all;

entity nand_or is
   port (a, b : in std_logic;
      c : out std_logic);
end;

architecture structure of nand_or is
   signal s1, s2 : std_logic;
begin
   u0: entity nand_2 port map (a, a, s1);
   u1: entity nand_2 port map (b, b, s2);
   u2: entity nand_2 port map (s1, s2, c);
end structure;
```

2.26

```
library ieee;
use ieee.std_logic_1164.all;

entity or_2 is
   port( i1, i2 : in std_logic;
      o1 : out std_logic
      );
end or_2;

architecture dataflow of or_2 is
begin
   o1 <= i1 or i2;
end dataflow;


library ieee;
use ieee.std_logic_1164.all;

entity invert is
   port( i1 : in std_logic;
      o1 : out std_logic
      );
end invert;

architecture dataflow of invert is
begin
   o1 <= not i1;
end dataflow;


library ieee;
use ieee.std_logic_1164.all;
```

```
entity ncs7sp19 is
   port(
      a : in std_logic;
      e_bar : in std_logic;
      y0 : out std_logic;
      y1 : out std_logic
      );
end ncs7sp19;

architecture structural of ncs7sp19 is
   signal s1 : std_logic ;
begin

   u1: entity or_2 port map (i1 => a, i2 => e_bar, o1 => y0);
   u2: entity invert port map (i1 => a, o1 => s1);
   u3: entity or_2 port map (i1 => s1, i2 => e_bar, o1 => y1);

end structural;
```

2.27

```
library ieee;
use ieee.std_logic_1164.all;

entity top_level is
  port(
       a : in std_logic;
       b : in std_logic;
       c : out std_logic
      );
end top_level;

architecture structural of top_level is
signal s1, s2, s3, s4 : std_logic ;
begin

 u1: entity not_1 port map (i1 => a, o1 => s1);
 u2: entity not_1 port map (i1 => b, o1 => s2);
 u3: entity and_2 port map (i1 => a, i2 => s2, o1 => s3);
 u4: entity and_2 port map (i1 => s1, i2 => b, o1 => s4);
 u5: entity or_2 port map (i1 => s3, i2 => s4, o1 => c);

end structural;
```

2.28 Only the code for the top-level design entity follows. The code for component design entities is the same as in the statement of Problem 2.24.

```
library ieee;
use ieee.std_logic_1164.all;

entity top_level is
   port(
      a, b, c : in std_logic;
      f : out std_logic
      );
end top_level;

architecture structural of top_level is
   signal s1, s2, s3 : std_logic;
begin

   u0: entity not_1 port map (i1 => b, o1 => s1);
   u1: entity or_2 port map   (i1 => a, i2 => s1, o1 => s2);
   u2: entity or_2 port map   (i1 => s1, i2 => c, o1 => s3);
   u3: entity and_2 port map   (i1 => s2, i2 => s3, o1 => f);

end structural;
```

2.29

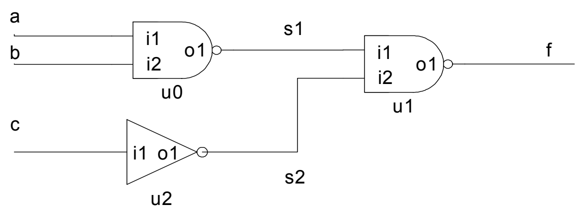
(a)

```
library ieee;
use ieee.std_logic_1164.all;

entity nand_2 is
   port(
      in1, in2 : in std_logic;
      out1 : out std_logic
      );
end nand_2;

architecture dataflow of nand_2 is
begin
   out1 <= in1 nand in2;
end dataflow;


library ieee;
use ieee.std_logic_1164.all;

entity nand_ckt is
   port(
      a : in std_logic;
      b : in std_logic;
      c_bar : in std_logic;
      f : out std_logic
      );
end nand_ckt;

architecture structural of nand_ckt is
   signal s1 : std_logic;
begin

   u1: entity nand_2 port map (in1 => a, in2 => b, out1 => s1);
   u2: entity nand_2 port map (in1 => s1, in2 => c_bar, out1 => f);

end structural;
```

(b)
```
architecture structural of nand_ckt is
   component nand_2 is
      port (in1, in2 : in std_logic;
         out1 : out std_logic);
   end component ;

   signal s1 : std_logic;
begin

   u1:  nand_2 port map (in1 => a, in2 => b, out1 => s1);
   u2:  nand_2 port map (in1 => s1, in2 => c_bar, out1 => f);

end structural;
```

(c)
```
library ieee;
use ieee.std_logic_1164.all;

entity nand_ckt is
 port(
  a : in std_logic;
  b : in std_logic;
  c_bar : in std_logic;
  f : out std_logic
  );
end nand_ckt;

architecture dataflow of nand_ckt is
 signal s1 : std_logic;
begin

 f <= (a nand b) nand c_bar;

end dataflow;
```

2.30



2.31 The RTL hierarchical view corresponds to the results from step 1 of the synthesis process and depends on the style of the description. The technology dependent flattened-to-gates view corresponds to the results from step 3.

2.32 When the same system is described using different coding styles, but synthesized to the same target PLD, we expect that the resulting synthesized logic at the gate level is identical for simple designs. However, for complex designs, the gate level logic, while functionally equivalent, may not be identical. This can occur because when starting from different functionally equivalent descriptions of the same system, the synthesizer optimization processes may not produce the same gate level results. Still, the results are functionally equivalent.

2.33 Rectangular blocks are used instead of distinctive shape logic symbols because in a structural style design what is being represented is the interconnection of design entities, not their functions.

2.34 The level of abstraction used in describing a system refers to the amount of implementation or structural detail in a description.

2.35 Use of a high level of abstraction allows us to describe a complex system's behavior without being overwhelmed by the large amount of detail and complexity required to describe the system at a much lower level of abstraction, such as the gate level.