the design. The WORK library is the default library, and this is the storage place for the current design you are working with and where that design is placed after it is compiled. The STD library is the storage place for the miscellaneous and logical operators such as **not**, **and**, and **nor**. The STD library also contains relational operators such as =, >, <. Refer to Section 4.10, VHDL Examples (at the end of this chapter), for a list of all the supported operators. The IEEE library is the storage place for 9-value data types called std_logic in the **package** IEEE. STD_LOGIC_1164. The main reason for using the IEEE library is for design portability—that is, you can use your VHDL source code with many different software vendors.

Listing 4.1 shows a **library clause** and a **use clause** for a VHDL design.

```
--These are the library and use clauses (The Library Part)
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

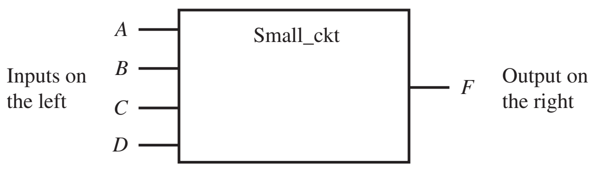
LISTING 4.1 Making the IEEE library and IEEE.STD_LOGIC_1164 package visible to a design

The **library** IEEE clause makes the IEEE library visible to a design. A library contains packages. The IEEE.STD_LOGIC_1164 package is specified by the **use clause** so that data type definitions, functions, and procedures that reside in the package are visible to the design. A specific component name can be used inside the package, but the wild card **ALL** is used to indicate that all the declarations inside the package can be used.

4.4 THE ENTITY DECLARATION

Think of a **black box** and inputs *A*, *B*, *C*, *D* (or terminal signals) and an output *F* as shown in Figure 4.3.

FIGURE 4.3 Black box for a design entity named Small_ckt



The term *black box* is used, as in circuit theory, to specify only the terminal or external signals for a design entity and not its contents. What is inside the box or the contents of the black box are not visible. A black box is equivalent to a symbol in a schematic. The terminal signals may represent the pins on an integrated circuit chip or the terminal signals of an embedded hardware block within a larger design. Small_ckt (short for Small_circuit) is simply the name or label that is used to identify a specific design entity.

Listing 4.2 shows the ED (entity declaration) for the black box in Figure 4.3.

LISTING 4.2 Entity declaration for black box named Small_ckt in Figure 4.3

```
--This is the entity declaration for design entity Small_ckt
entity Small_ckt is port (
    A, B, C, D : in std_logic;
    F : out std_logic
    );
end Small_ckt;
```

Things you should notice about the VHDL entity declaration in Listing 4.2:

- Comments start with `--`.
- **Keywords** (reserved words) are shown in **boldface type** in the VHDL code. Refer to Section 4.10, VHDL Examples (at the end of this chapter), for a list of all keywords that are supported for synthesis.
- An ED must start with the keyword **entity**.
- VHDL is not case sensitive and has a free format, which means that there is no formatting convention for spacing and indentations.
- Signal names (such as A, B, etc.) and labels (such as Small_ckt) have the following rules: (a) the first character must be a letter, (b) numbers may be included as well as the underscore character (_), (c) no adjacent underscore characters may be used, (d) an underscore character may not be used as the last character, and (e) spaces are not allowed. Keywords (reserved words) cannot be used as signal names or labels. Signal names and labels are formally called **identifiers**.
- Input signals are mode **in**.
- Std_logic is the **data type** used in most digital designs [this is a 9-value system of which only the values 0, L (weak 0—pull-down resistor), 1, H (weak 1—pull-up resistor), Z (high impedance—tri-state), and - (don't care) are supported for synthesis, i.e., hardware implementations]. Note: Values L, H, and Z are case sensitive; that is, they must be uppercase. Just for your information, the values U (uninitialized), W (weak unknown), and X (forcing unknown) are not supported by synthesis. This leaves only six values that are supported for synthesis. The values U, W, and X are only supported for simulations of designs on a computer but not for synthesizing designs via hardware components. The Xilinx compiler supports the value - (don't care), but not all compilers do.
- Output signals are mode **out**.
- Observe where semicolons are required and where a semicolon is not allowed, that is, before ");" in the **entity**.
- A port must open with a left parenthesis and end with a right parenthesis.
- The entity must terminate with the keyword **end**.

Data Type Bit

A signal can be assigned the data type "bit." The data type "bit" can only take on the values of 1 and 0. This limits the usefulness of this data type, because it cannot have the values L, H, Z, or -, which are the additional data types used by data type "std_logic." Also, VHDL has strict data type checking so that one data type cannot be assigned to a different data type. Data type "std_logic" is the preferred data type for most designs because it has six values that are supported for synthesis rather than the data type "bit," which only has two values.

4.5 THE ARCHITECTURE DECLARATION

The contents of a black box represent the actual digital circuit for the design. In VHDL, the actual digital circuit is expressed in an **architecture declaration**. Figure 4.4 shows the contents of the black box labeled Small_ckt.

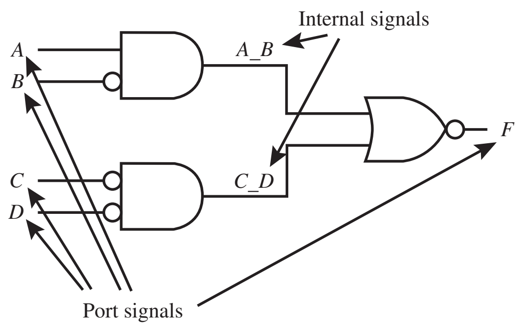


FIGURE 4.4 Digital circuit: contents of the black box for design entity named Small_ckt in Figure 4.3

VHDL has three design styles available for use in an architecture declaration. We will use each of these design styles for the digital circuit in Figure 4.4 to help you to understand their syntax. The three design styles are listed as follows:

1. Dataflow design style.
2. Behavioral design style.
3. Structural design style.

4.5.1 Comments about a Dataflow Design Style

Only (1) *Boolean equations*, (2) *conditional signal assignments (CSAs)*, and (3) *selected signal assignments (SSAs)* can be used in a dataflow design style. These signal assignments are **concurrent statements (CSs)** because they are evaluated by the VHDL compiler concurrently or at the same time. The order in which they are written is not important. The simplest form of a dataflow design style is a Boolean equation. The equation implies how the hardware should be created; therefore, **implication** is used to create the hardware required for a circuit when using a dataflow design style.

4.5.2 Comments about a Behavioral Design Style

Only (1) *Boolean equations*, (2) *if statements*, and (3) *case statements* can be used in a behavioral design style. These equations and statements must be placed inside a **process**. The complete **process** is a **concurrent statement**; however, the statements inside the **process** are evaluated sequentially by the VHDL compiler—that is, in the order in which they are written in the process. A behavioral architecture declaration creates the structure for a circuit by **inference** (creating the circuit from deduction by reasoning from the general to the specific).

4.5.3 Comments about a Structural Design Style

Hardware blocks or components are used in a structural design style. An **annotated circuit** or **schematic** must be provided to use this design style—that is, a schematic with all the input, output, and internal signals clearly labeled. The schematic is separated into hardware blocks or components and then simply connected together just like wiring a digital circuit using gates. The installation or placement of the hardware blocks or components are referred to in VHDL as **instantiation** and their interconnections are referred to as **port mapping**. Instantiation and port-mapping statements are **concurrent statements** because they are evaluated at the same time. The order in which they are written is not important. A structural architecture declaration creates the structure for a circuit by the way you **wire it up or connect the components**.

As you will see later, a combination of design styles can also be used together in an architecture declaration to generate a hardware circuit for a **system**. We will refer to a collection of modules or components that form a hardware circuit in VHDL as a system. When we use a combination of design styles, we will call the name of the architecture **mixed** to indicate that a mixture of design styles is being used.

4.6 DATAFLOW DESIGN STYLE

Listing 4.3 shows an AD (architecture declaration) with a dataflow design style for the digital circuit in Figure 4.4—that is, the design entity named Small_ckt.

```
--This is a dataflow AD for Small_ckt
architecture dataflow of Small_ckt is
begin
    F <= (A and not B) nor (not C and not D);
end dataflow;
```

LISTING 4.3
Dataflow architecture declaration with a Boolean equation

Things you should notice about the VHDL architecture declaration in Listing 4.3:

- The architecture declaration must start with the keyword **architecture**.
- The architecture declaration must reference the name of the design entity—that is, Small_ckt for this example.
- The keyword **begin** is required before the **simple signal assignment statement** for *F*—that is, a **Boolean equation**. A single Boolean equation is sufficient; however, multiple Boolean equations could be specified if we add a declaration for the internal signals *A_B* and *C_D*.
- A Boolean expression is assigned to the signal *F* via the signal assignment symbol (<=).
- A semicolon is required to terminate the **simple signal assignment statement**.
- VHDL has no order of precedence for logic operators with two signals (or operands) such as **and** and **nor;** therefore, parentheses must be used to establish precedence for these logic operators. **Not** has the highest precedence of all the operators in VHDL and therefore does not need parentheses—that is, the result for **not** *B* and **not** (*B*) are the same.
- The dataflow AD must terminate with the keyword **end**.

If there is more than one signal assignment statement between **begin** and **end**, all of the signal assignment statements are evaluated at the same time.

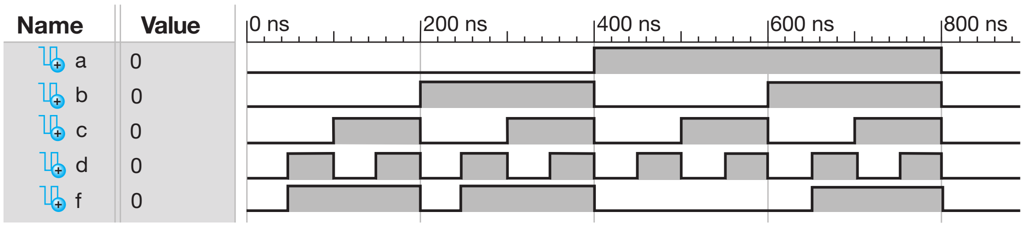
Complete VHDL code for the design entity Small_ckt is obtained by combining Listings 4.1, 4.2, and 4.3 as shown in Listing 4.4.

```
--These are the library and use clauses
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--This is the entity declaration for Small_ckt
entity Small_ckt is port (
    A, B, C, D : in std_logic;
    F : out std_logic
    );
end Small_ckt;

--This is a dataflow architecture declaration for Small_ckt
architecture dataflow of Small_ckt is
begin
    F <= (A and not B) nor (not C and not D);
end dataflow;
```

LISTING 4.4
Complete VHDL design entity for Small_ckt using a dataflow architecture declaration with a Boolean equation (project: Small_ckt_Bool)

To check for VHDL design correctness means to check for correct functionality. Waveform 4.1 shows a simulation waveform diagram with the correct functionality of design entity Small_ckt. This can easily be confirmed by creating a truth table for function F and then checking the waveform diagram with the truth table output.

WAVEFORM 4.1 Simulation for the correct functionality of design entity Small_ckt



A different architecture declaration can be used with the same entity declaration to implement the same hardware block. Rather than a **Boolean equation**, two other types of signal assignment statements may be used in a dataflow architecture declaration. A **conditional signal assignment (CSA)** is one type. Another type is a **selected signal assignment (SSA)**.

Listing 4.5 shows a complete VHDL design entity for Small_ckt using a conditional signal assignment in an architecture declaration.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Small_ckt is port (
     A, B, C, D : in std_logic;
     F : out std_logic
     );
end Small_ckt;

architecture dataflow of Small_ckt is
begin
     --writing F in terms of its 1s
     F <= '1' when ((A and not B) nor (not C and not D)) = '1' else
           '0';
--or one of the following:
     --writing F in terms of its 0s, observe that nor must be changed to or
--   F <= '0' when ((A and not B) or (not C and not D)) = '1' else
--         '1';
     --writing F in terms of its 1s
--   F <= '1' when (A and not B) = '1' nor (not C and not D) = '1' else
--         '0';
     --writing F in terms of its 1s
--   F <= '1' when (A = '1' and B = '0') nor (C = '0' and D = '0') else
--         '0';
end dataflow;
```

LISTING 4.5 Complete VHDL design entity for Small_ckt using a dataflow architecture declaration with a conditional signal assignment (project: Small_ckt_CSA)

Things you should notice about the VHDL design in Listing 4.5:

- Only one signal assignment symbol (<=) is used for a conditional signal assignment.
- A single bit is represented as '1' or '0'—that is, using single quotation marks.
- A Boolean expression or signal may be compared to a bit using the "=" relational operator. Parentheses must be used around a Boolean expression when it is compared to a bit using a relational operator.
- Parentheses determine the order of the precedence for logic operators.
- A semicolon is required to terminate a conditional signal assignment.
- The **when** conditions in a conditional signal assignment are prioritized. The first output statement listed (*F* <= '1') has the highest priority and will be executed first, if the condi-

tion is true. If the condition is not true, the first output statement is skipped and the second output statement (*F* <= '0') will be executed next, and so on (for additional conditions).

- If a conditional signal assignment does not have the last **else** (the **else** after the last **when**), a bad thing can happen: the circuit is not implemented correctly, because one or more inputs are not used. The output of the circuit is tied to GND or V_{CC} making the circuit nonfunctional or useless. For combinational circuits, be sure to include the last **else** (the **else** after the last **when)** so that latches or nonfunctional circuits are not inadvertently created on the outputs of the circuits.
- The simulation for the VHDL design in Listing 4.5 is the same as Waveform 4.1 shown earlier.

Listing 4.6 shows a complete VHDL design entity for Small_ckt using a selected signal assignment (SSA).

LISTING 4.6
Complete VHDL design entity for Small_ckt using a dataflow architecture declaration with a selected signal assignment (project: Small_ckt_SSA)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Small_ckt is port (
     A, B, C, D : in std_logic;
     F : out std_logic
     );
end Small_ckt;

architecture dataflow of Small_ckt is
begin
     with (A and not B) nor (not C and not D) select
          F <= '1' when '1',
               '0' when '0',
               '0' when others;
end dataflow;
```

Things you should notice about the VHDL design in Listing 4.6:

- The evaluation of a Boolean expression provides the selection value.
- Parentheses determine the order of precedence for logic operators.
- Only one signal assignment symbol (<=) is used for a selected signal assignment.
- A single bit is represented as '1' or '0'—that is, using single quotation marks.
- The selected signal assignment examines the value of the expression and executes only the signal assignment that matches the **when** value, and all the others signal assignments are skipped.
- **When others**; is required to terminate a selected signal assignment statement. **When others** is used to ensure that all possible select values for the expression (*A* **and not** *B*) **nor** (**not** *C* **and not** *D*) are included in the choice list after **when**. The std_logic values that must be included for the expression are 0, L, 1, H, Z, and -. Note that the single dash "-" represents a don't care in VHDL.
- Observe where commas are required in a selected signal assignment.
- The conditions for a selected signal assignment are not prioritized like a conditional signal assignment and thus require conditions that do not overlap or are mutually exclusive values.
- Order is not important, and the only output statement that is executed is the one that meets the condition.
- The simulation for the VHDL design in Listing 4.6 is the same as Waveform 4.1 shown earlier.

Listing 4.4, 4.5, or 4.6 may be used for the hardware design of the digital circuit in Figure 4.4 using a dataflow architecture declaration. You should now be very familiar with how to create VHDL design entities using a dataflow architecture declaration.

4.7 BEHAVIORAL DESIGN STYLE

Statements *within* a process—that is, in the body of the process (between **begin** and **end process**)—are evaluated in the order they are written (one after the other or sequentially), which is similar to the evaluation of statements in normal software programming languages such as Pascal, C, Java, Perl, and Ruby. Even though a process contains statements that are evaluated sequentially, a complete **process statement**, from **process** through **end process**, is a concurrent statement. This means that a complete process statement is evaluated concurrently with another complete process statement or with other concurrent statements in an architecture declaration. The fact that a complete process statement is a concurrent statement might not seem important to you at this time, but it will be very important later, so keep this in mind.

We will now show how to implement the design entity named Small_ckt using behavioral architecture declarations. To refresh your memory, Figures 4.3 and 4.4 are combined in Figure 4.5 to provide you with a handy reference.

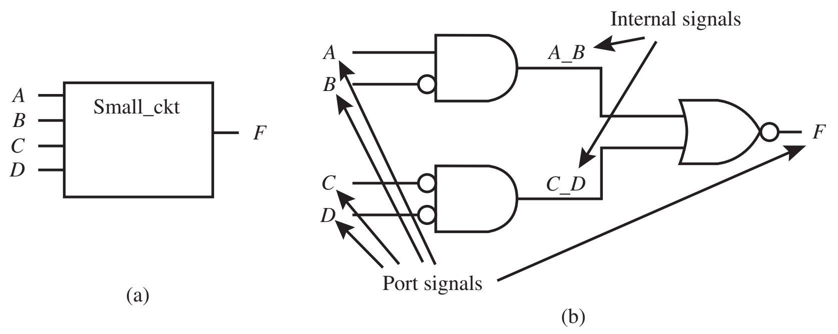


FIGURE 4.5 Design entity Small_ckt (a) black box, and (b) contents of black box

The VHDL design entity for a behavioral architecture declaration must have the library clause, the use clause, and the entity declaration as shown in Listing 4.7 for design entity Small_ckt.

LISTING 4.7 Library clause, use clause, and entity declaration for Small_ckt

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Small_ckt is port (
    A, B, C, D : in std_logic;
    F : out std_logic
    );
end Small_ckt;
```

Listing 4.8 shows a behavioral architecture declaration using a process with Boolean equations, for design entity Small_ckt.

LISTING 4.8 Behavioral architecture declaration for design entity Small_ckt using a process with Boolean equations

```
architecture behavioral of Small_ckt is
    --internal signal declarations for A_B and C_D
    signal A_B, C_D: std_logic;
begin
example: process (A, B, C, D, A_B, C_D)
begin
    A_B <= A and not B;
    C_D <= not C and not D;
    F <= A_B nor C_D;
end process example;
end behavioral;
```

Things you should notice about the VHDL design in Listing 4.8:

- Internal signals (such as *A_B* and *C_D*) must be declared before the first **begin**, because it is illegal in VHDL to declare signals in a process. These internal signals are only visible inside the architecture.
- A process statement is used after the first **begin**. An optional name such as "example:" (notice that it must be followed by a colon) may be used to name a process. The optional name may or may not be included following **end process** (this is your choice).
- The process statement contains a list of signals called the **sensitivity list**. The process wakes up or executes when an event occurs for any one of the signals *A*, *B*, *C*, *D*, *A_B*, and *C_D* in the sensitivity list—that is, when a signal in the sensitivity list changes from 1 to 0 or from 0 to 1. When a process wakes up, it executes once and then suspends (stops or does nothing) and waits for the next event to occur.
- All the inputs must be included in the sensitivity list, including the internal signals, or the process will not work properly.
- The keyword **is** may be used after the sensitivity list but is practically never used because it is optional.
- A process has a **begin** and an **end** just like an architecture.
- Each statement in a process ends with a semicolon.
- Each statement in a process is executed in the order that it is written in the process. You may consider that the process executes in zero time each time a signal in the sensitivity list changes. Writing the statements in a different order may result in a different design functionality. A simulation run can be made to verify proper design functionality.
- Using intermediate simple signal assignments in a process tends to show the circuit layout.

Note that Boolean equations *can* be used inside a process; however, conditional signal assignments (CSAs) and selected signal assignments (SSAs) *cannot* be used inside a process.

Complete VHDL code for the design entity Small_ckt using a behavioral architecture declaration with Boolean equations is obtained by combining the Listings for 4.7 and 4.8 (project: Small_ckt_Proc_Bool). The simulation for the VHDL design obtained by combining Listings 4.7 and 4.8 is the same as Waveform 4.1 shown earlier.

Listing 4.9 shows a behavioral architecture declaration using a process with an **if–then–else statement** for design entity Small_ckt.

```
architecture behavioral of Small_ckt is
begin
process (A, B, C, D)
begin
    if (A = '1' and B = '0') nor (C = '0' and D = '0') then F <= '1';
    else F <= '0';
    end if;
end process;
end behavioral;
```

LISTING 4.9 Behavioral architecture declaration for design entity Small_ckt using a process with an if–then–else statement

Things you should notice about the VHDL design in Listing 4.9:

- A process statement is used after the first **begin**. Notice that the process was not named as it was in Listing 4.8. Remember, naming a process is optional.
- The process statement contains a list of signals called the sensitivity list. The process wakes up or executes when an event occurs—that is, when a signal in the sensitivity list changes

from 1 to 0 or from 0 to 1 (any one of the signals *A*, *B*, *C*, and *D* in the list). When a process wakes up, it executes once and then suspends and waits for the next event to occur.

- A process declaration has a **begin** and an **end** just like an architecture declaration.
- Semicolons are required after the simple signal assignment statements *F* <= '1', *F* <= '0' and also after **end if** in the process.
- The conditions in the if–then–else statements are prioritized. The first output statement listed (*F* <= '1') has the highest priority and will be executed first, if the condition is true. If the condition is not true, the first output statement is skipped and the second output statement (*F* <= '0') will be executed next, and so on (for additional conditions).
- If an if–then–else statement does not have the last **else** (the **else** after the last **then**), a couple of bad things can happen: the circuit is not combinational, because the output is latched, or the circuit is not implemented correctly, because one or more inputs are not used. In the first case, the output now contains a memory and is no longer a combinational circuit. In the second case, the output of the circuit is tied to GND or V_{CC}, making the circuit nonfunctional or useless. For combinational circuits, be sure to include the last **else** (the **else** after the last **then**) so that latches or nonfunctional circuits are not inadvertently created for the outputs of the circuits.
- An if–then–else statement must be placed in a process between **begin** and **end process**.
- You may observe that the if–then–else statement is similar to the conditional signal assignment (CSA).

Complete VHDL code for the design entity Small_ckt using a behavioral architecture declaration with a process and an if–then–else statement, is obtained by combining Listings 4.7 and 4.9 (project: Small_ckt_Proc_if). The simulation for the VHDL design obtained by combining Listings 4.7 and 4.9 is the same as Waveform 4.1 shown earlier.

To handle multiple conditions, if–then–else statements can be nested (a statement within a statement). Listing 4.10 shows the code for two conditions using nested if–then–else statements.

LISTING 4.10
Nested if–then–else statements (project: example_Proc_if)

```
if x = '1' then f <= a;
else if y = '1' then f <= b;
     else f <= c;
     end if;
end if;
```

Output *F* is prioritized such that its output is *A* if *X* is 1, *B* if *X* is 0 and *Y* is 1, and *C* if *X* is 0 and *Y* is 0. The condition for signal *X* has priority over the signal *Y*. When both signal *X* and signal *Y* are inactive (or 0), then *F* is *C*.

Things you should notice about the VHDL if statements in Listing 4.10:

- Nested if–then–else statements require an **end if** to terminate each **if**. If six conditions were required then six **end ifs** would be required.
- For readability, line up each **if**, **else**, and **end if**.

The truth table for Listing 4.10 is shown in Table 4.1.

TABLE 4.1 Truth table for Listing 4.10

X	*Y*	*F*	
1	1	*A*	Highest priority
1	0	*A*	↓
0	1	*B*	
0	0	*C*	Lowest priority

To simplify nesting, it is more efficient to use an **if–then–elsif statement**. Listing 4.11 shows equivalent code as Listing 4.10 using an if–then–elsif statement.

```
if x = '1' then f <= a;
elsif y = '1' then f <= b;
else f <= c;
end if;
```

LISTING 4.11
Equivalent code as in Listing 4.10 using an if–then–elsif statement (project: example_Proc_elsif)

Things you should notice about the VHDL if statement in Listing 4.11:

- Using an if–then–elsif statement results in fewer lines of code (only one less line for this simple case).
- Only one **end if** is required to terminate an if–then–elsif statement. If six conditions were required then only one **end if** would be required when **elsif** is used rather than **else if** for each condition after the first **if**.
- For readability, line up the single **if**, all the **elsif**s, the final **else**, and the single **end if**. Be sure to use the final **else** to make the circuit combinational and not sequential. If you leave off the final **else** and compile the design by running Synthesize – XST, you will get the warning: "Found 1-bit latch for signal <f>". When you generate a latch in this manner, your circuit may have timing problems. So if you get this warning, fix your VHDL code by writing complete if statements.

A **case statement** is often preferred to nested **if statements**. This is because nested if statements and elsif statements can result in more logic gates, because the conditions are prioritized. Listing 4.12 shows a behavioral architecture declaration with a process and a case statement for design entity Small_ckt.

```
architecture behavioral of Small_ckt is
begin
process (A, B, C, D)
begin
    case (A and not B) nor (not C and not D) is
        when '1' => F <= '1';
        when '0' => F <= '0';
        when others => null;
    end case;
end process;
end behavioral;
```

LISTING 4.12
Behavioral architecture declaration for design entity Small_ckt with a process and a case statement

Things you should notice about the VHDL architecture declaration in Listing 4.12:

- A case statement must be placed in a process between **begin** and **end process**.
- The case statement examines the value of the signal that is listed after **case**—for example, (*A* **and not** *B*) **nor** (**not** *C* **and not** *D*)—and only executes the output statement to the right of the symbol => that matches the **when** value for the signal and all the other output statements are skipped.
- The conditions for a case statement are not prioritized like an if statement and thus require conditions that do not overlap or are mutually exclusive values.
- **When others** is used to ensure that all possible select values for the signal listed after **case** are included in the choice list after **when**. The std_logic values that must be included for the signal are 0, L, 1, H, Z, and -. Note that the single dash "-" represents a don't care in VHDL. The keyword **null** means do nothing—that is, perform no action.

- Order is not important, and the only output statement that is executed is the one that meets the condition.
- You may observe that the case statement is similar to the selected signal assignment (SSA).
- Observe that a semicolon is required after each signal assignment and also after the keyword **null** to terminate the case statement.

Listing 4.13 is alternate code for design entity Small_ckt with a behavioral architecture declaration with a process, Boolean equation, and a case statement:

LISTING 4.13
Behavioral architecture declaration for design entity Small_ckt with a process, a Boolean equation, and a case statement

```
architecture behavioral of Small_ckt is
     --internal signal declaration for sel (or select)
     signal sel: std_logic;
begin
process (A, B, C, D, sel)
begin
     sel <= (A and not B) nor (not C and not D);
     case sel is
          when '1' => F <= '1';
          when '0' => F <= '0';
          when others => null;
     end case;
end process;
end behavioral;
```

Things you should notice about the VHDL architecture declaration in Listing 4.13:

- The signal declaration statement for sel (or select) must be placed in the signal declaration part of the architecture, which is between **architecture** and the first **begin**. This is an internal signal, so a mode (such as **in**, **out**, or **inout**) must not be assigned.
- To use *SEL* in the process, it must be included in the sensitivity list.
- The internal signal *SEL* is assigned its value in the process after **begin** using a simple signal assignment (Boolean equation).

Complete VHDL code for the design entity Small_ckt using a behavioral architecture declaration with a case statement is obtained by combining Listings 4.7 and 4.12 (project: Small_ckt_Proc_case) or Listings 4.7 and 4.13 (project: Small_ckt_Proc_case_alt). The simulations for the VHDL designs obtained by combining Listings 4.7 and 4.12 and obtained by combining Listings 4.7 and 4.13 are the same as Waveform 4.1 shown earlier.

4.8 STRUCTURAL DESIGN STYLE

A structural architecture declaration is often referred to as a **hierarchal design approach**. The procedure we use to obtain a structural design is listed as follows:

Step 1: Partition the design into manageable hardware blocks called components.
Step 2: Write complete VHDL code to define each component with a dataflow design style or a behavioral design style.
Step 3: Write the library part and the entity declaration for the top level of the structural design.
Step 4: Declare all the internal signals and all the components between **architecture** and the first **begin** in the architecture declaration for the top level of the structural design.
Step 5: Instantiate the components (or place and connect the components) after the first **begin** in the architecture declaration for the top level of the structural design.

Step 1: Figure 4.6 shows the design entity Small_ckt partitioned or subdivided into smaller units called components.

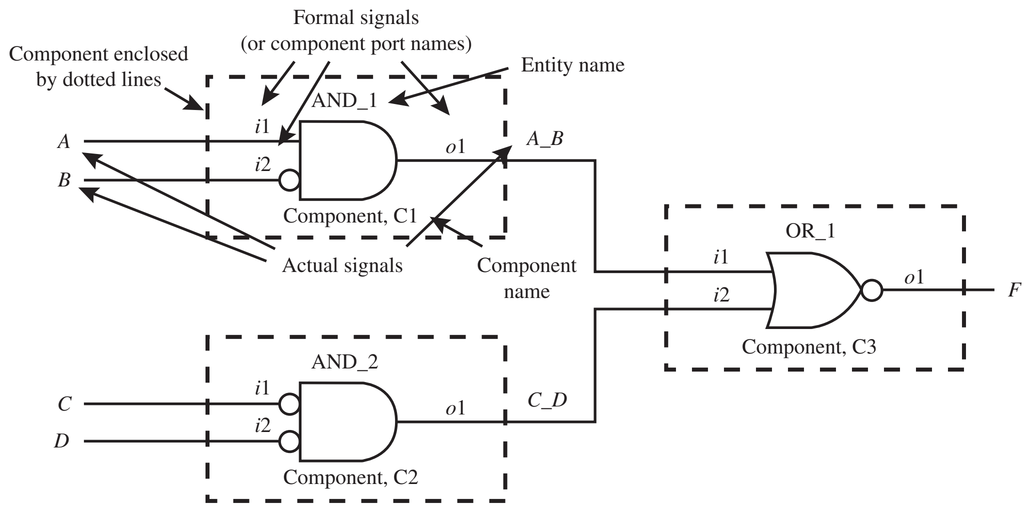


FIGURE 4.6 Design entity Small_ckt partitioned or subdivided into smaller units called components

Things you should notice about the design entity Small_ckt in Figure 4.6:

- The components are enclosed by dotted lines.
- The components are given the arbitrary entity names AND_1, AND_2, and OR_1. These names (or identifiers) are used as the names of the entities when writing the definitions for the components in step 2.
- The input and output signals of the components (component port names) are called **formal signals** and are given the arbitrary signal names *i*1, *i*2, and *o*1. Formal signals must be used when writing the definitions for the components in step 2.
- The components are given the arbitrary component names C1, C2, and C3. These names (or identifiers) are used when instantiating the components in step 5.

Step 2: Listing 4.14 shows complete VHDL code to define the components AND_1, AND_2, and OR_1. Each design should be simulated to verify correct functionality.

LISTING 4.14 Component definitions for AND_1, AND_2, and OR_1

```
--Component definition for AND_1
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity and_1 is port (
     i1, i2: in std_logic;
     o1: out std_logic
     );
end and_1;

architecture dataflow of and_1 is
begin
     o1 <= i1 and not i2;
end dataflow;

--Component definition for AND_2
library IEEE;
use IEEE.STD_LOGIC_1164.all;
```

(Continued)

```
entity and_2 is port (
     i1, i2: in std_logic;
     o1: out std_logic
     );
end and_2;

architecture dataflow of and_2 is
begin
     o1 <= not i1 and not i2;
end dataflow;

--Component definition for OR_1
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity or_1 is port (
     i1, i2: in std_logic;
     o1: out std_logic
     );
end or_1;

architecture dataflow of or_1 is
begin
     o1 <= i1 nor i2;
end dataflow;
```

Things you should notice about the VHDL component definitions in Listing 4.14:

- Complete VHDL code is written for each component definition.
- The entity name for each component is AND_1, AND_2, and OR_1, respectively.
- The formal signals (or component port names) *i*1 (short for input1), *i*2, and *o*1 (short for output1) must be used to define the components.

Step 3: Listing 4.15 shows the library part and the entity declaration for the top level of the structural design for design entity Small_ckt.

LISTING 4.15
Library part and entity declaration for design entity Small_ckt

```
--Structural Design (top level)
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Small_ckt is port (
     A, B, C, D : in std_logic;
     F : out std_logic
     );
end Small_ckt;
```

Things you should notice about the VHDL code for design entity Small_ckt in Listing 4.15:

- The name of the entity is Small_ckt, which represents the complete design or complete system.
- The signals in the entity Small_ckt are the external input signals and the external output signals for the complete design or complete system.

Steps 4 and 5: Listing 4.16 shows the **internal signal declarations** and the **component declarations**, which are placed between **architecture** and the first **begin** in the architecture declaration for the structural design—that is, design entity Small_ckt. Listing 4.16 also shows the **component instantiations**, which are placed after the first **begin**.

```
architecture structural of Small_ckt is
--internal signal declarations for A_B and C_D
     signal A_B, C_D: std_logic;
--Component declaration for and_1
component and_1 is port (
     i1, i2: in std_logic;
     o1: out std_logic
     );
end component;
--Component declaration for and_2
component and_2 is port (
     i1, i2: in std_logic;
     o1: out std_logic
     );
end component;
--Component declaration for or_1
component or_1 is port (
     i1, i2: in std_logic;
     o1: out std_logic
     );
end component;
begin
--Component placement and connections (formally called component instantiations)
     C1: and_1 port map (i1 => A, i2 => B, o1 => A_B);
     C2: and_2 port map (i1 => C, i2 => D, o1 => C_D);
     C3: or_1 port map (i1 => A_B, i2 => C_D, o1 => F);
end structural;
```

LISTING 4.16 Structural architecture declaration for design entity Small_ckt

Things you should notice about the VHDL architecture declaration in Listing 4.16:

- The signal declaration **signal** *A_B*, *C_D*: std_logic; is required to declare two internal signals *A_B* and *C_D*, which are the output signals of the two AND gates in Figure 4.6. Signals *A_B* and *C_D* are only visible inside the architecture declaration of Small_ckt. A mode (**in** or **out**) is not required for internal signals. Internal signal declarations must always be placed between **architecture** and the first **begin**.
- The component declarations are placed between **architecture** and the first **begin**.
- Notice that a component declaration has exactly the same information as the entity declaration for the component only it starts with the word **component** and is terminated by **end component**.
- Each component must be placed and connected in the architecture of the structural design. This is done via the **instantiation statement** C1: and_1 **port map** (*i*1 => *A*, *i*2 => *B*, *o*1 => *A_B*); for component 1, C2: and_2 **port map** (*i*1 => *C*, *i*2 => *D*, *o*1 => *C_D*); for component 2, and C3: or_1 **port map** (*i*1 => *A_B*, *i*2 => *C_D*, *o*1 => *F*); for component 3. This method for connecting the components is called **name association**, and has the form **formal signal** => **actual signal**.
- An alternate method for connecting components is called **positional association**. Positional association for C1 is written as C1: and_1 **port map** (*A*, *B*, *A_B*). The actual signals must be placed in *exactly* the same positions as they are listed in the component declaration. Name association is highly recommended over positional association, because it is easier to make mistakes when using positional association.

- The order of writing the instantiation statements is not important because these are concurrent statements.
- The structural architecture declaration must be terminated by the keyword **end**.

It should be noted that instantiation statements (or component instantiations) *cannot* be placed inside a process.

Complete VHDL code for the design entity Small_ckt using a structural architecture declaration is obtained by combining Listings 4.14, 4.15, and 4.16. The VHDL code for the component definitions must be placed in the same project as the VHDL code for the structural design—that is, design entity Small_ckt—to allow the VHDL code for the structural design to correctly compile. Listing 4.17 shows the complete VHDL code for the design entity Small_ckt, including the definitions of the components.

LISTING 4.17
Complete VHDL code for the design entity Small_ckt using a structural architecture declaration, including the definitions of the components (project: Small_ckt_structural).

```
--Component definition for AND_1
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity and_1 is port (
     i1, i2: in std_logic;
     o1: out std_logic
     );
end and_1;
architecture dataflow of and_1 is
begin
     o1 <= i1 and not i2;
end dataflow;
--Component definition for AND_2
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity and_2 is port (
     i1, i2: in std_logic;
     o1: out std_logic
     );
end and_2;
architecture dataflow of and_2 is
begin
     o1 <= not i1 and not i2;
end dataflow;
--Component definition for OR_1
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity or_1 is port (
     i1, i2: in std_logic;
     o1: out std_logic
     );
end or_1;
architecture dataflow of or_1 is
begin
     o1 <= i1 nor i2;
end dataflow;
--Structural Design (top level)
library IEEE;
use IEEE.STD_LOGIC_1164.all;
```

```
entity Small_ckt is port (
     A, B, C, D : in std_logic;
     F : out std_logic
     );
end Small_ckt;
architecture structural of Small_ckt is
     signal A_B, C_D: std_logic;
component and_1 is port (
     i1, i2: in std_logic;
     o1: out std_logic
     );
end component;
component and_2 is port (
     i1, i2: in std_logic;
     o1: out std_logic
     );
end component;
component or_1 is port (
     i1, i2: in std_logic;
     o1: out std_logic
     );
end component;
begin
     C1: and_1 port map (i1 => A, i2 => B, o1 => A_B);
     C2: and_2 port map (i1 => C, i2 => D, o1 => C_D);
     C3: or_1 port map (i1 => A_B, i2 => C_D, o1 => F);
end structural;
```

The simulation for the VHDL design in Listing 4.17 is the same as Waveform 4.1 shown earlier.

As we have pointed out, the structural design style can be treated as a hierarchal design approach by considering the components as the lower levels and the architecture declaration as the top level. In practice, we partition the design into the desired components (the lower levels of the hierarchy), write the definitions for the components, simulate each component to verify that it works as expected, and then write the architecture declaration for the structure (the top level of the hierarchy). The top level can then be simulated or run in hardware to verify that it works as expected.

In Chapter 2 you were introduced to a **flat design approach**, where each of the modules in the system was included within a single architecture declaration. Renaming component C1 as module 1, component C2 as module 2, and component C3 as module 3 in the circuit in Figure 4.6, we can write VHDL code for a flat design approach as shown in Listing 4.18 using a dataflow design style for each module.

LISTING 4.18 Flat design approach for the circuit in Figure 4.6 using a dataflow design style for each module (project: Small_ckt_flat)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Small_ckt_flat is port (
     a,b,c,d : in std_logic;
     f : out std_logic
     );
end Small_ckt_flat;
```

(Continued)

```
architecture dataflow of Small_ckt_flat is
     signal a_b, c_d : std_logic;
begin
--Module 1, AND_1
     a_b <= a and not b;
--Module 2, AND_2
     c_d <= not c and not d;
--Module 3, OR_1
     f <= a_b nor c_d;
end dataflow;
```

The simulation for the VHDL design in Listing 4.18 is the same as Waveform 4.1 shown earlier.

Observe how much simpler it is to use a flat design approach compared to a hierarchal design approach (or structural design style) for the simple circuit in Figure 4.6. By simpler, we mean fewer lines of code. For medium design projects, one may elect to use a hierarchal or a flat design approach. For a very large design project, a hierarchal design approach is very important because this approach allows individuals in a group to work on a portion of the design in a real-world situation.

In a very large design project, a flat design approach suffers because it may be hard to understand conceptually and possibly hard to modify.

4.9 IMPLEMENTING WITH WIRES AND BUSES

A **wire** and a **bus** are hardware terms. A wire carries a single bit of information. A bus, which represents a collection of wires, carries multiple bits of information. A signal on a wire and on a bus is shown in Figure 4.7.

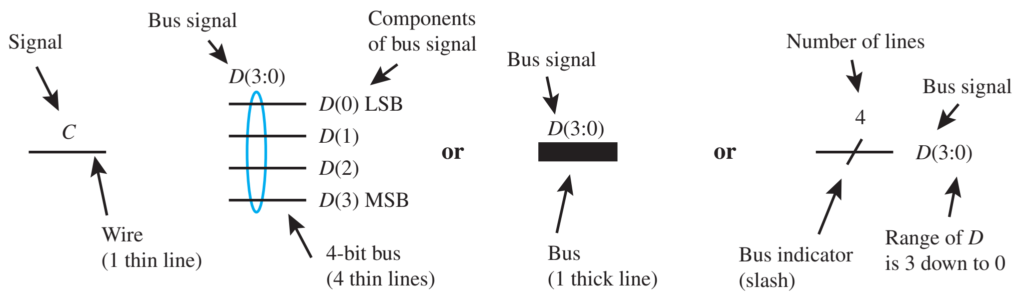


FIGURE 4.7 Signal on a wire and on a bus

In VHDL, signal *C* has a **data type** called std_logic that we have been using throughout this chapter. A nonbus signal or a signal for a single wire or line such as signal *C* may be referred to as a **scalar**. Bus signal *D*(3:0) has a **data type** called std_logic_vector (3 **downto** 0), where *D*(3) is the MSB (most significant bit) and *D*(0) is the LSB (least significant bit). The components of a bus signal *D*(3:0) in VHDL may not be written as *D*3, *D*2, *D*1, and *D*0. A bus signal or a signal for a number of lines such as signal *D*(3:0), with the range 3 down to 0, may be referred to as a **vector**.

If the bus signal in Figure 4.7 were rewritten as *D*(0:3), then *D*(0) is the MSB and *D*(3) is the LSB. In VHDL, the bus signal *D*(0:3) would be the std_logic_vector (0 **to** 3) data type. When writing VHDL code to represent hardware blocks or design entities, you must choose the correct data type for each wire and bus in the design.

A black box for a 3-to-8 decoder is shown in Figure 4.8, drawn in two different but equivalent ways. The first drawing uses thin lines to represent the individual bus wires. The second drawing uses slashes with the number of lines to represent the buses.

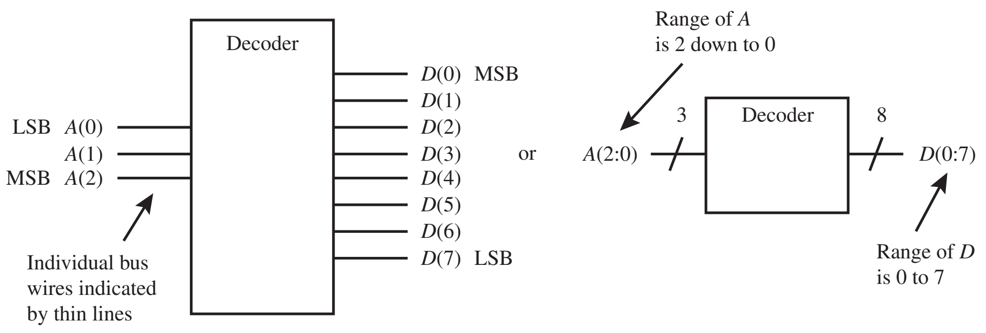


FIGURE 4.8 Black box for 3-to-8 decoder

Table 4.2 shows the truth table for the decoder.

TABLE 4.2 Truth table for 3-to-8 decoder

A(2)	*A*(1)	*A*(0)	*D*(0)	*D*(1)	*D*(2)	*D*(3)	*D*(4)	*D*(5)	*D*(6)	*D*(7)
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

A behavioral architecture declaration using a case statement is shown in Listing 4.19 for the 3-to-8 decoder. The bit pattern (or code) for bus *A*(2:0) is binary while the bit pattern for *D*(0:7) is one-hot. For a one-hot output code, only one output bit is **hot** or **active**—that is, 1—while all other output bits are inactive or 0.

```
architecture behavional of Decoder is
begin
process (A)
begin
     case A is
          when "000" => D <= "10000000";
          when "001" => D <= "01000000";
          when "010" => D <= "00100000";
          when "011" => D <= "00010000";
          when "100" => D <= "00001000";
          when "101" => D <= "00000100";
          when "110" => D <= "00000010";
          when "111" => D <= "00000001";
          when others => null;
     end case;
end process;
end behavioral;
```

LISTING 4.19 Behavioral architecture declaration for design entity Decoder using a case statement

Things you should notice about the VHDL architecture declaration in Listing 4.19:

- Signal *A* must be read by the process, and so signal *A* must be in the sensitivity list of the process.
- Signal *A* represents a single signal consisting of 3 bits (a bus). A string of bits—that is, more than one bit—must be included in double quotation marks. Recall that a single bit is included in single quotation marks. The data type for bus signal *A* is std_logic_vector (2 **downto** 0). This must be the data type listed in the entity declaration for the signal *A*.
- Signal *D* represents a single signal consisting of 8 bits (a bus). A string of bits—that is, more than one bit—must be included in double quotation marks. The data type for bus signal *D* is std_logic_vector (0 **to** 7). This must be the data type listed in the entity declaration for the signal *D*.
- A **when others clause** is required at the end of the choice list of the case statement. **When others** is used to ensure that all possible select values for signal *A* are included in the choice list after **when**. The std_logic values that must be included for signal *A* are 0, L, 1, H, Z, and -. Note that the single dash "-" represents a don't care in VHDL. The keyword **null** means do nothing—that is, perform no action.

The library clause, use clause, and entity declaration for the design entity Decoder are shown in Listing 4.20.

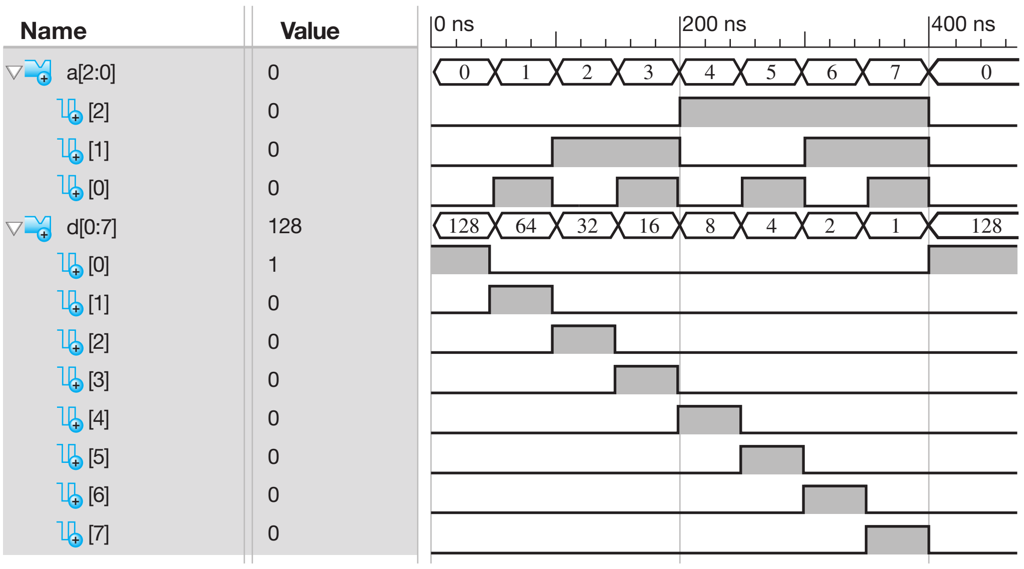
LISTING 4.20
Library clause, use clause, and entity declaration for design entity Decoder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Decoder is port (
     A : in std_logic_vector (2 downto 0);
     D : out std_logic_vector (0 to 7)
     );
end Decoder;
```

To obtain complete VHDL code for the design entity Decoder, combine the listings for 4.19 and 4.20 (project: Decoder_3t8_Proc_case).

The waveforms shown in Waveform 4.2 represent the correct functionality of design entity Decoder. This can easily be confirmed by comparing Waveform 4.2 with the truth table for the decoder in Table 4.2.

WAVEFORM 4.2
Simulation for the correct functionality of design entity Decoder (project: Decoder_3t8_Proc_case)


Name	Value
a[2:0]	0
[2]	0
[1]	0
[0]	0
d[0:7]	128
[0]	1
[1]	0
[2]	0
[3]	0
[4]	0
[5]	0
[6]	0
[7]	0

Rather than declaring a bus in the entity for signal *A* as shown in Listing 4.20, individual signals *A*2, *A*1, and *A*0 can be declared and then grouped together to form a bus using an expression called an **aggregate**. An aggregate is a collection or group of elements. The VHDL code in Listing 4.21 shows how to form a bus signal *A* with the aggregate (*A*2, *A*1, *A*0). The bus signal *A* is needed for the case statement in the design.

LISTING 4.21
Forming a bus with an aggregate (project: Decoder_3t8_Proc_case_agg)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Decoder is port (
     A2,A1,A0 : in std_logic;
     D : out std_logic_vector (0 to 7)
     );
end Decoder;
architecture behavioral of Decoder is
     signal A: std_logic_vector (2 downto 0);
begin
     A <= (A2,A1,A0);
process (A)
begin
     case A is
          when "000" => D <= "10000000";
          when "001" => D <= "01000000";
          when "010" => D <= "00100000";
          when "011" => D <= "00010000";
          when "100" => D <= "00001000";
          when "101" => D <= "00000100";
          when "110" => D <= "00000010";
          when "111" => D <= "00000001";
          when others => null;
     end case;
end process;
end behavioral;
```

Things you should notice about the VHDL code in Listing 4.21:

- The data type for signals *A*2, *A*1, and *A*0 is std_logic as declared in the entity.
- An internal signal *A* is declared as std_logic_vector (2 downto 0) and placed between **architecture** and the first **begin**, because a bus signal is needed for the case statement.
- An aggregate (*A*2, *A*1, *A*0) is assigned to the internal signal *A* and placed in the architecture after the first **begin**.
- The bus signal *A* is now available for use in the case statement.

The simulation for the VHDL design in Listing 4.21 is the same as Waveform 4.2 shown earlier.

Forming a bus with an aggregate is a handy concept to know. The placement of the elements (or the order of the elements) in the aggregate is very important. The left-most element in the list is the MSB, while the right-most element in the list is the LSB. In Listing 4.21, making the assignment *A* <= (*A*0, *A*1, *A*2) would be an error for the logic circuit, because *A* is defined as a std_logic_vector (2 downto 0), assuming that *A*2(MSB), *A*1, *A*0(LSB).

An aggregate can also be used to form a larger bus using the individual elements of the bus to form the aggregate. Do not use a vector in an aggregate; however, you may use the individual elements to form a larger bus with a larger range. Suppose we wanted to form the bus *A*(4:0). If some of the values of the individual elements of the bus [say, *A*(4) and *A*(3)] must be added to *A*(2 downto 0) to form the larger bus then the correct way to write the aggregate is *A* <= (*A*(4), *A*(3), *A*(2), *A*(1), *A*(0)). Writing the aggregate as *A* <= (*A*(4), *A*(3), *A*(2 **downto** 0)) would be a syntax error. If you want to form a larger bus with vectors you may use the concatenation operator "&" as follows: *A* <= (*A*(4) & *A*(3) & *A*(2 **downto** 0)). Just like the aggregate, the order in which the elements are placed in the list is very important—that is, the left-most element in the list is the MSB and the right-most element in the list is the LSB.

4.10 VHDL EXAMPLES

Figure 4.9 shows an alphabetical list of **keywords** that support synthesis.

FIGURE 4.9 An alphabetical list of keywords that support synthesis

A	**abs, all, alias, and, architecture, array, attribute**
B	**begin, block, body, buffer**
C	**case, component, configuration, constant**
D	**downto**
E	**else, elsif, end, entity, exit**
F	**for, function**
G	**generate, generic, group**
I	**if, in, inout, is**
L	**library, literal, loop**
M	**map, mod**
N	**nand, next, nor, not, null**
O	**of, or, others, out**
P	**package, port, procedure, process**
R	**range, record, rem, return, rol, ror**
S	**select, signal, sla, sll, sra, srl, subtype**
T	**then, to, type**
U	**until, use**
V	**variable**
W	**wait, when, while, with**
X	**xnor, xor**

Figure 4.10 shows a list of **supported operators**.

FIGURE 4.10 A list of supported operators

Group	Supported Operators
1. Logical	(**and, or, nand, nor, xor, xnor**)
2. Relational	(=, /=, <, <=, >, >=)
3. Shifting	(**sll, srl, sla, sra, rol, ror**)
4. Adding	(+, −, &)
5. Unary signing	(+, −)
6. Multiplying	(*, /, **mod, rem**)
7. Miscellaneous	(**, **abs, not**)

Note: Without parentheses, operators in group 7 have the highest precedence, followed by group 6, and so on. down to the lowest precedence, or group 1. Within each group, there is no operator precedence, and precedence must be established by parentheses. In an expression without parentheses, operators are applied left to right.

4.10.1 Design with Scalar Inputs and Outputs

Figure 4.11 shows the design entity for an OR gate called OR_2. This design uses scalar inputs and outputs.

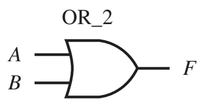


FIGURE 4.11 Design entity for an OR gate called OR_2

The library part, entity declaration, and partial architecture declaration for design entity OR_2 are shown in Listing 4.22.

LISTING 4.22 The library part, entity declaration, and partial architecture declaration for design entity OR_2

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity OR_2 is port (
     A, B : in std_logic;
     F  : out std_logic
     );
end OR_2;

architecture design_style of OR_2 is
begin
     <Architecture body>
end design_style;
```

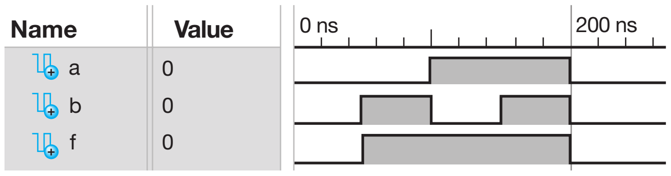
Any one of the architecture design styles presented in Listing 4.23 can be used to complete the partial architecture declaration in Listing 4.22. These design styles use scalar inputs and outputs.

LISTING 4.23 Architecture design styles with scalar inputs and outputs

```
Boolean equation:
   --place the Boolean equation between begin and end process for
   --a behavioral design
      F <= A or B;

Conditional signal assignment (CSA):
   --Note: parentheses are required around A or B
      F <= '1' when (A or B) = '1' else
            '0';
   Note: CSA is the concurrent equivalent of the if statement.

Selected signal assignment (SSA):
      with A or B select
         F <= '1' when '1',
               '0' when '0',
               '0' when others;
   Note: SSA is the concurrent equivalent of the case statement.

If statement:
   --place the if statement between begin and end process
   --Note: parentheses are required around A or B
      if (A or B) = '1' then F <= '1';
      else F <= '0';
      end if;
   Note: if statement is the sequential equivalent of the CSA.
                                                  (Continued)
```

```
Case statement:
   --place the case statement between begin and end process
      case A or B is
         when '1' => F <= '1';
         when '0' => F <= '0';
         when others => null;
      end case;
   Note: case statement is the sequential equivalent of the SSA.
```

The waveform diagram shown in Waveform 4.3 represents the correct functionality of design entity OR_2.



WAVEFORM 4.3 Simulation for the correct functionality of design entity OR_2 (project: or_2_Bool)

All of the architecture design styles presented in Listing 4.23 provide the same simulation as shown in Waveform 4.3.

4.10.2 Design with Vector Inputs and Outputs

Figure 4.12 shows the design entity for a 2-to-4 decoder called DEC2_4. This design uses vector inputs and outputs.

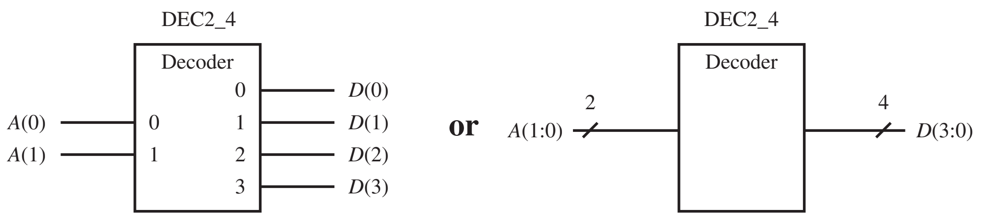


FIGURE 4.12 Design entity for a 2-to-4 decoder called DEC2_4

The library part, entity declaration, and partial architecture declaration for design entity DEC2_4 are shown in Listing 4.24.

LISTING 4.24 The library part, entity declaration, and partial architecture declaration for design entity DEC2_4

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity DEC2_4 is port (
     A : in std_logic_vector (1 downto 0);
     D : out std_logic_vector (3 downto 0)
     );
end DEC2_4;

architecture design_style of DEC2_4 is
begin
     <Architecture body>
end design_style;
```

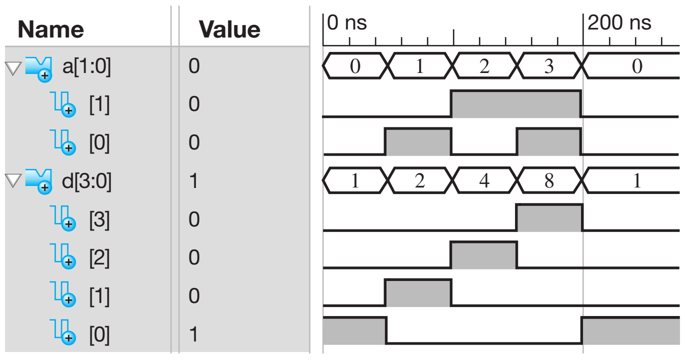
Any one of the architecture design styles presented in Listing 4.25 can be used to complete the partial architecture declaration in Listing 4.24. These design styles use vector inputs and outputs.

LISTING 4.25
Architecture design styles with vector inputs and outputs

```
Boolean equations:
      D(0) <= not A(1) and not A(0);
      D(1) <= not A(1) and A(0);
      D(2) <= A(1) and not A(0);
      D(3) <= A(1) and A(0);

Conditional signal assignment (CSA):
      D <= "0001" when A = "00" else
           "0010" when A = "01" else
           "0100" when A = "10" else
           "1000";
   Note: CSA is the concurrent equivalent of the if statement.

Selected signal assignment (SSA):
      with A select
         D <= "0001" when "00",
              "0010" when "01",
              "0100" when "10",
              "1000" when "11",
              "0001" when others;
   Note: SSA is the concurrent equivalent of the case statement.

If statement:
   --place the if statement between begin and end process
      if    A = "00" then D <= "0001";
      elsif A = "01" then D <= "0010";
      elsif A = "10" then D <= "0100";
      else                D <= "1000";
      end if;
   Note: if statement is the sequential equivalent of the CSA.

Case statement:
   --place the case statement between begin and end process
      case A is
         when "00" => D <= "0001";
         when "01" => D <= "0010";
         when "10" => D <= "0100";
         when "11" => D <= "1000";
         when others => null;
      end case;
   Note: case statement is the sequential equivalent of the SSA.
```

The waveform diagram shown in Waveform 4.4 represents the correct functionality of design entity DEC2_4.

WAVEFORM 4.4 Simulation for the correct functionality of design entity DEC2_4 (project: DEC2_4_Bool)



All of the architecture design styles presented in Listing 4.25 provide the same simulation as shown in Waveform 4.4.

4.10.3 Common VHDL Constructs

Listing 4.26 is a brief list of common VHDL constructs that we have used in this chapter. Other constructs exist and are available by clicking on the light bulb icon (language templates) in Xilinx ISE Projector Navigator.

LISTING 4.26 A brief list of common VHDL constructs

```
   Library Part:
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

   Entity Declaration:
entity <Entity name> is port (
       <Port name> : <Mode> <Type>;
       <Other ports>. . .
       );
end <Entity name>;

   Architecture Declaration:
architecture <Architecture name> of <Entity name> is
   --Signal_and_component declarations (internal signal
   --declarations, component declarations)
begin
    <Architecture body>
end <Architecture name>;

   Boolean equation:
<Signal name> <= <Expression>;

   Conditional signal assignment (CSA):
<Signal name> <= <Expression> when <Condition> else
                 <Expression> when <Condition> else
                 <Expression>;

   Selected signal assignment (SSA):
with <Choice expression> select
     <Signal name> <= <Expression> when <Choices>,
                      <Expression> when <Choices>,
                      <Expression> when others;
   Process Declaration:
process (<All input signals separated by commas>)
begin
   <Boolean equation, if statement, case statement >;
end process;

   If statement:
if <Condition> then <Statement>;
elsif <Condition> then <Statement>;
else <Statement>;
end if;

   Case statement:
case (<Signal name>) is
   when ". . .00" => <Statement>;
   when ". . .01" => <Statement>;
   when ". . .10" => <Statement>;
   when ". . .11" => <Statement>;
   . . .
   when others => <Statement>;
end case;

   Component declaration:
component <Component name> port (
   <Port name> : <Mode> <Type>;
   <Other ports>. . .
   );
end component;

   Component instantiation:
<Instance name> : <Component name> port map (<Port name> =>
                                           <Signal name>,
                                           <Other ports>. . .);
```

PROBLEMS

Section 4.2 VHDL

4.1 What is a design entity?

4.2 What are the names of the three parts of a design entity in VHDL?

4.3 What part of a design entity for VHDL has data type definitions, functions, and procedures?

4.4 What part of a design entity for VHDL specifies the interface or the external inputs and outputs of a digital circuit?

4.5 What part of a design entity for VHDL specifies the functional composition or functionality of a digital circuit?

Section 4.3 The Library Part

4.6 What two clauses are required for the library part of a design entity in VHDL?

4.7 What two libraries are implicitly declared (built in) for VHDL designs?

4.8 What library must be added to a VHDL design to make it visible to the design?

4.9 Where is your design placed after it is compiled?

4.10 What library stores the logical and relational operators?

4.11 What library stores the 9-value data type called std_logic?

4.12 What is the main reason for using the IEEE library?

Section 4.4 The Entity Declaration

4.13 What can an entity declaration be thought of, in terms of circuit theory?

4.14 What are the three main features of a black box?

4.15 When writing VHDL, how do you insert a comment in the code?