GLOBAL EDITION

Digital Fundamentals

ELEVENTH EDITION

Thomas L. Floyd

ALWAYS LEARNING

PEARSON

Eleventh Edition | Global Edition

Digital Fundamentals

Thomas L. Floyd

PEARSON

Boston Columbus Indianapolis New York San Francisco Hoboken
Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montreal Toronto
Delhi Mexico City São Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo

Signals

In VHDL, signals are analogous to wires that interconnect components on a circuit board. The signals in Figure 5–39 are named OUT1 and OUT2. Signals are the *internal* connections in the logic circuit and are treated differently than the inputs and outputs. Whereas the inputs and outputs are declared in the entity declaration using the port statement, the signals are declared within the architecture using the signal statement. **Signal** is a VHDL keyword.

The Program

The program for the logic in Figure 5–39 begins with an entity declaration as follows:

```
entity AND_OR_Logic is
  port (IN1, IN2, IN3, IN4: in bit; OUT3: out bit);
end entity AND_OR_Logic;
```

The architecture declaration contains the component declarations for the AND gate and the OR gate, the signal definitions, and the component instantiations.

```
architecture LogicOperation of AND_OR_Logic is

  component AND_gate is
    port (A, B: in bit; X: out bit);         <-- Component declaration for the AND gate
  end component AND_gate;

  component OR_gate is
    port (A, B: in bit; X: out bit);         <-- Component declaration for the OR gate
  end component OR_gate;

  signal OUT1, OUT2: bit;                    <-- Signal declaration

begin
  G1: AND_gate port map (A => IN1, B => IN2, X => OUT1);
  G2: AND_gate port map (A => IN3, B => IN4, X => OUT2);    <-- Component instantiations describe
  G3: OR_gate port map (A => OUT1, B => OUT2, X => OUT3);       how the three gates are connected.
end architecture LogicOperation;
```

Component Instantiations

Let's look at the component instantiations. First, notice that the component instantiations appear between the keyword **begin** and the **end architecture** statement. For each instantiation an identifier is defined, such as G1, G2, and G3 in this case. Then the component name is specified. The keyword **port map** essentially makes all the connections for the logic function using the operator =>. For example, the first instantiation,

```
G1: AND_gate port map (A => IN1, B => IN2, X => OUT1);
```

can be explained as follows: *Input A of AND gate G1 is connected to input IN1, input B of the gate is connected to input IN2, and the output X of the gate is connected to the signal OUT1.*

The three instantiation statements together completely describe the logic circuit in Figure 5–39, as illustrated in Figure 5–40.

Although the data flow approach using Boolean expressions would have been easier and probably the best way to describe this particular circuit, we have used this simple circuit to explain the concept of the structural approach. Example 5–16 compares the structural and data flow approaches to writing a VHDL program for an SOP logic circuit.

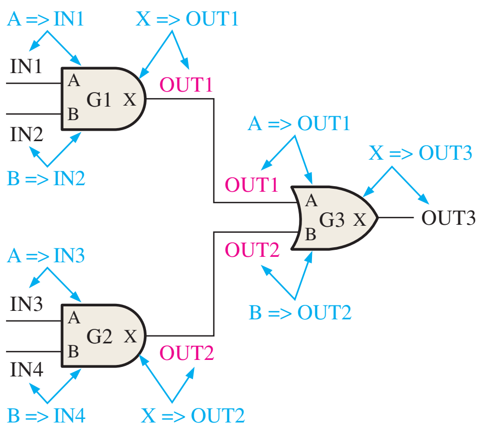


FIGURE 5–40 Illustration of the instantiation statements and port mapping applied to the AND-OR logic. Signals are shown in red.

EXAMPLE 5–16

Write a VHDL program for the SOP logic circuit in Figure 5–41 using the structural approach and compare with the data flow approach. Assume that VHDL components for a 3-input NAND gate and for a 2-input NAND are available. Notice the NAND gate G4 is shown as a negative-OR.

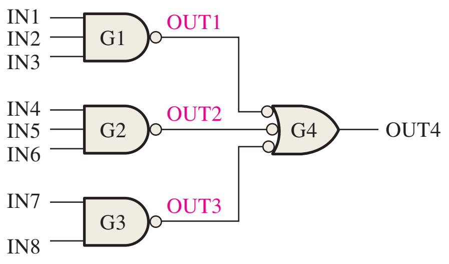


FIGURE 5–41

Solution

The structural approach:

The components and component instantiations are highlighted. Lines preceded by two hyphens are comment lines and are not part of the program.

```
--Program for the logic circuit in Figure 5–41

entity SOP_Logic is
  port (IN1, IN2, IN3, IN4, IN5, IN6, IN7, IN8: in bit; OUT4: out bit);
end entity SOP_Logic;

architecture LogicOperation of SOP_Logic is

--component declaration for 3-input NAND gate

  component NAND_gate3 is
    port (A, B, C: in bit X: out bit);
  end component NAND_gate3;

--component declaration for 2-input NAND gate

  component NAND_gate2 is
    port (A, B: in bit; X: out bit);
  end component NAND_gate2;

  signal OUT1, OUT2, OUT3: bit;
```

```
begin
G1: NAND_gate3 port map   (A => IN1, B => IN2, C => IN3, X => OUT1);
G2: NAND_gate3 port map   (A => IN4, B => IN5, C => IN6, X => OUT2);
G3: NAND_gate2 port map   (A => IN7, B => IN8, X => OUT3);
G4: NAND_gate3 port map   (A => OUT1, B => OUT2, C => OUT3, X => OUT4);
end architecture LogicOperation;
```

The data flow approach:

The program for the logic circuit in Figure 5–41 using the data flow approach is written as follows:

```
entity SOP_Logic is
  port (IN1, IN2, IN3, IN4, IN5, IN6, IN7, IN8: in bit; OUT4: out bit);
end entity SOP_Logic;

architecture LogicOperation of SOP_Logic is
begin
  OUT4 <= (IN1 and IN2 and IN3) or (IN4 and IN5 and IN6) or (IN7 and IN8);
end architecture LogicOperation;
```

As you can see, the data flow approach results in a much simpler code for this particular logic function. However, in situations where a logic function consists of many blocks of complex logic, the structural approach might have an advantage over the data flow approach.

Related Problem

If another NAND gate is added to the circuit in Figure 5–41 with inputs IN9 and IN10, write a component instantiation to add to the program.

Applying Software Development Tools

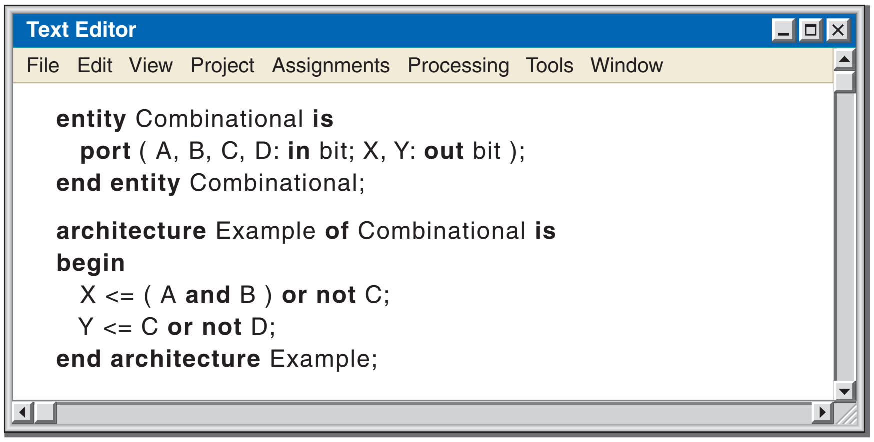
A software development package must be used to implement an HDL design in a target device. Once the logic has been described using an HDL and entered via a software tool called a code or text editor, it can be tested using a simulation to verify that it performs properly before actually programming the target device. Using software development tools allows for the design, development, and testing of combinational logic before it is committed to hardware.

Typical software development tools allow you to input VHDL code on a text-based editor specific to the particular development tool that you are using. The VHDL code for a combinational logic circuit has been written using a text-based editor for illustration and appears on the computer screen as shown in Figure 5–42. Many code editors provide enhanced features such as the highlighting of keywords.

After the program has been written into the text editor, it is passed to the compiler. The compiler takes the high-level VHDL code and converts it into a file that can be downloaded to the target device. Once the program has been compiled, you can create a simulation for testing. Simulated input values are inserted into the logic design and allow for verification of the output(s).

You specify the input waveforms on a software tool called a waveform editor, as shown in Figure 5–43. The output waveforms are generated by a simulation of the VHDL code that you entered on the text editor in Figure 5–42. The waveform simulation provides the resulting outputs X and Y for the inputs A, B, C, and D in all sixteen combinations from $0\,0\,0\,0_2$ to $1\,1\,1\,1_2$.

Recall from Chapter 3 that there are several performance characteristics of logic circuits to be considered in the creation of any digital system. Propagation delay, for example, determines the speed or frequency at which a logic circuit can operate. A timing simulation can be used to mimic the propagation delay through the logic design in the target device.



```
entity Combinational is
  port ( A, B, C, D: in bit; X, Y: out bit );
end entity Combinational;

architecture Example of Combinational is
begin
  X <= ( A and B ) or not C;
  Y <= C or not D;
end architecture Example;
```

FIGURE 5–42 A VHDL program for a combinational logic circuit after entry on a generic text editor screen that is part of a software development tool.

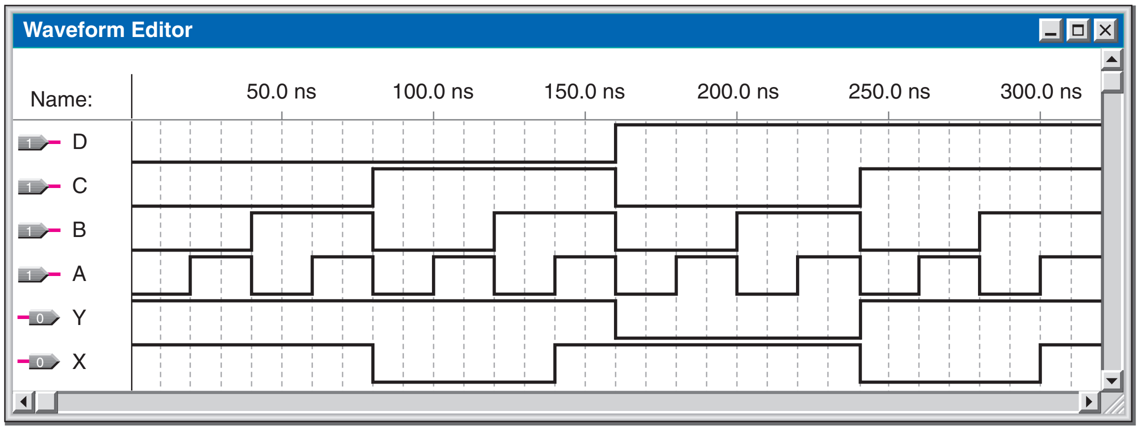


FIGURE 5–43 A typical waveform editor tool showing the simulated waveforms for the logic circuit described by the VHDL code in Figure 5–42.

SECTION 5–6 CHECKUP

1. What is a VHDL component?
2. State the purpose of a component instantiation in a program architecture.
3. How are interconnections made between components in VHDL?
4. The use of components in a VHDL program represents what approach?

5–7 Troubleshooting

The preceding sections have given you some insight into the operation of combinational logic circuits and the relationships of inputs and outputs. This type of understanding is essential when you troubleshoot digital circuits because you must know what logic levels or waveforms to look for throughout the circuit for a given set of input conditions.

In this section, an oscilloscope is used to troubleshoot a fixed-function logic circuit when a device output is connected to several device inputs. Also, an example of signal tracing and waveform analysis methods is presented using a scope or logic analyzer for locating a fault in a combinational logic circuit.

After completing this section, you should be able to

- Define a circuit node
- Use an oscilloscope to find a faulty circuit node
- Use an oscilloscope to find an open input or output
- Use an oscilloscope to find a shorted input or output
- Discuss how to use an oscilloscope or a logic analyzer for signal tracing in a combinational logic circuit

In a combinational logic circuit, the output of a driving device may be connected to two or more load devices as shown in Figure 5–44. The interconnecting paths share a common electrical point known as a **node**.

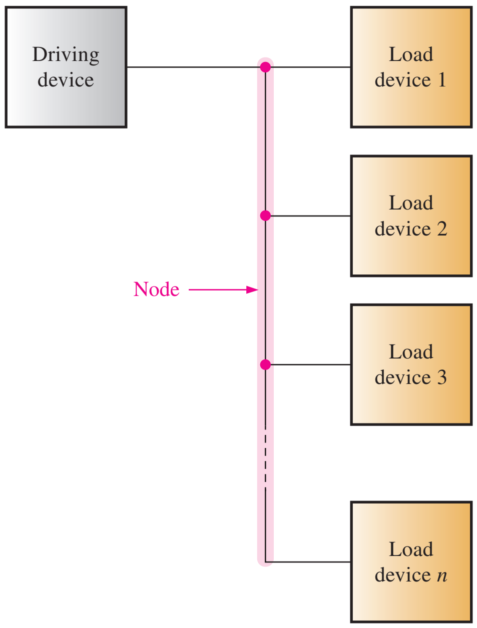


FIGURE 5–44 Illustration of a node in a logic circuit.

The driving device in Figure 5–44 is driving the node, and the other devices represent loads connected to the node. A driving device can drive a number of load device inputs up to its specified fan-out. Several types of failures are possible in this situation. Some of these failure modes are difficult to isolate to a single bad device because all the devices connected to the node are affected. Common types of failures are the following:

1. ***Open output in driving device.*** This failure will cause a loss of signal to all load devices.
2. ***Open input in a load device.*** This failure will not affect the operation of any of the other devices connected to the node, but it will result in loss of signal output from the faulty device.
3. ***Shorted output in driving device.*** This failure can cause the node to be stuck in the LOW state (short to ground) or in the HIGH state (short to V_{CC}).
4. ***Shorted input in a load device.*** This failure can also cause the node to be stuck in the LOW state (short to ground) or in the HIGH state (short to V_{CC}).

Troubleshooting Common Faults

Open Output in Driving Device

In this situation there is no pulse activity on the node. With circuit power on, an open node will normally result in a “floating” level, as illustrated in Figure 5–45.

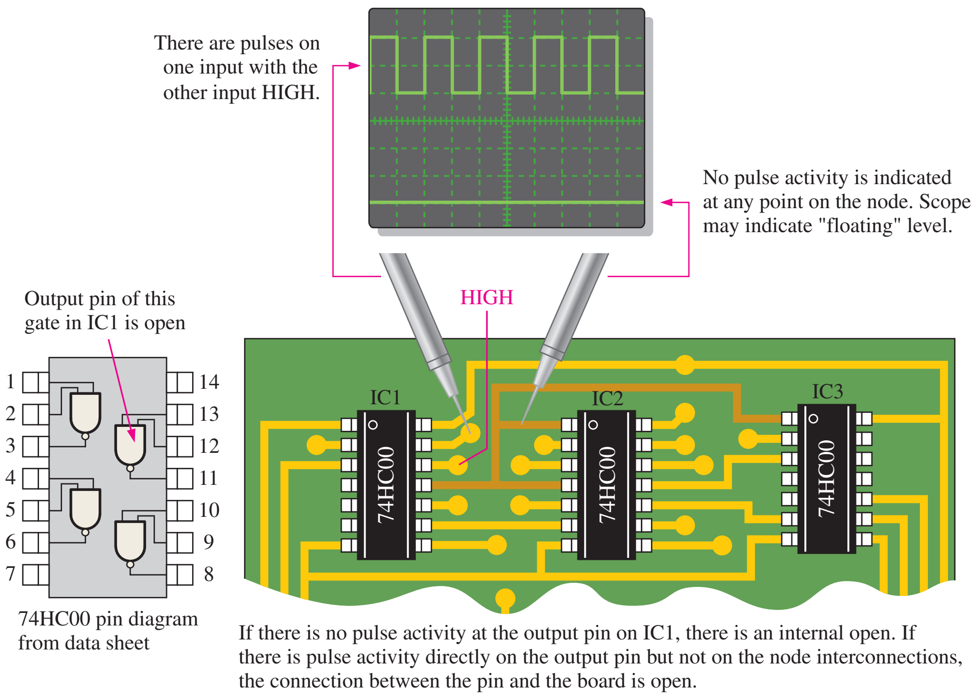

There are pulses on one input with the other input HIGH.
No pulse activity is indicated at any point on the node. Scope may indicate "floating" level.
Output pin of this gate in IC1 is open
HIGH
IC1
IC2
IC3
74HC00
74HC00
74HC00
1
2
3
4
5
6
7
14
13
12
11
10
9
8
74HC00 pin diagram from data sheet
If there is no pulse activity at the output pin on IC1, there is an internal open. If there is pulse activity directly on the output pin but not on the node interconnections, the connection between the pin and the board is open.

FIGURE 5–45 Open output in driving device. Assume a HIGH is on one input.

Open Input in a Load Device

If the check for an open driver output in IC1 is negative (there is pulse activity), then a check for an open input in a load device should be performed. Check the output of each device for pulse activity, as illustrated in Figure 5–46. If one of the inputs that is normally connected to the node is open, no pulses will be detected on that device’s output.

Output or Input Shorted to Ground

When the output is shorted to ground in the driving device or the input to a load device is shorted to ground, it will cause the node to be stuck LOW, as previously mentioned. A quick check with a scope probe will indicate this, as shown in Figure 5–47. A short to ground in the driving device’s output or in any load input will cause this symptom, and further checks must therefore be made to isolate the short to a particular device.

Signal Tracing and Waveform Analysis

Although the methods of isolating an open or a short at a node point are useful from time to time, a more general troubleshooting technique called **signal tracing** is of value in just

HandsOnTip

When troubleshooting logic circuits, begin with a visual check, looking for obvious problems. In addition to components, visual inspection should include connectors. Edge connectors are frequently used to bring power, ground, and signals to a circuit board. The mating surfaces of the connector need to be clean and have a good mechanical fit. A dirty connector can cause intermittent or complete failure of the circuit. Edge connectors can be cleaned with a common pencil eraser and wiped clean with a Q-tip soaked in alcohol. Also, all connectors should be checked for loose-fitting pins.