Chapter 3: Modeling Concurrent Functionality

This chapter presents a set of built-in operators that will allow logic to be modeled within the VHDL architecture. This chapter then presents a series of combinational logic model examples.

Learning Outcomes—After completing this chapter, you will be able to:

3.1 Describe the various built-in operators within VHDL.
3.2 Design a VHDL model for a combinational logic circuit using concurrent signal assignments and logical operators.
3.3 Design a VHDL model for a combinational logic circuit using conditional signal assignments.
3.4 Design a VHDL model for a combinational logic circuit using selected signal assignments.
3.5 Design a VHDL model for a combinational logic circuit that contains delay.

3.1 VHDL Operators

There are a variety of pre-defined operators in the IEEE standard package. It is important to note that operators are defined to work on specific data types and that not all operators are synthesizable. It is also important to remember that VHDL is a hardware description language, not a programming language. In a programming language, the lines of code are executed sequentially as they appear in the source file. In VHDL, the lines of code represent the behavior of real hardware. As a result, all signal assignments are by default executed concurrently unless specifically noted otherwise. All operations in VHDL must be on like types, and the result must be assigned to the same type as the operation inputs.

3.1.1 Assignment Operator

VHDL uses **<=** for all signal assignments and **:=** for all variable and initialization assignments. These assignment operators work on all data types. The target of the assignment goes on the left of these operators and the input arguments go on the right.

Example:

```
F1 <= A;          -- F1 and A must be the same size and type
F2 <= '0';        -- F2 is type bit in this example
F3 <= "0000";     -- F3 is type bit_vector(3 downto 0) in this example
F4 <= "hello";    -- F4 is type string in this example
F5 <= 3.14;       -- F5 is type real in this example
F6 <= x"1A";      -- F6 is type bit_vector(7 downto 0), x"1A" is in HEX
```


B. J. LaMeres, *Quick Start Guide to VHDL*, https://doi.org/10.1007/978-3-030-04516-6_3

3.1.2 Logical Operators

VHDL contains the following logical operators:

Operator	Operation
not	Logical negation
and	Logical AND
nand	Logical NAND
or	Logical OR
nor	Logical NOR
xor	Logical Exclusive-OR
xnor	Logical Exclusive-NOR

These operators work on types bit, bit_vector, and boolean. For operations on the type bit_vector, the input vectors must be the same size and will take place in a bit-wise fashion. For example, if two 8-bit buses called BusA and BusB were AND'd together, BusA(0) would be individually AND'd with BusB(0), BusA(1) would be individually AND'd with BusB(1), etc. The not operator is a unary operation (i.e., it operates on a single input), and the keyword is put before the signal being operated on. All other operators have two or more inputs and are placed in-between the input names.

Example:

```
F1 <= not A;
F2 <= B and C;
```

The order of precedence in VHDL is different from in Boolean algebra. The NOT operator is a higher priority than all other operators. All other logical operators have the same priority and have no inherent precedence. This means that in VHDL, the AND operator will *not* precede the OR operation as it does in Boolean algebra. Parentheses are used to explicitly describe precedence. If operators are used that have the same priority and parentheses are not provided, then the operations will take place on the signals listed first moving left to right in the signal assignment. The following are examples on how to use these operators:

Example:

```
F3 <= not D nand E;       -- D will be complemented first, the result
                          -- will then be NAND'd with E, then the
                          -- result will be assigned to F3
F4 <= not (F or G);       -- the parentheses take precedence so
                          -- F will be OR'd with G first, then
                          -- complemented, and then assigned to F4.

F5 <= H nor I nor J;      -- logic operations can have any number of
                          -- inputs.

F6 <= K xor L xnor M;     -- XOR and XNOR have the same priority so with
                          -- no parentheses given, the logic operations
                          -- will take place on the signals from
                          -- left to right. K will be XOR'd with L first,
                          -- then the result will be XNOR'd with M.
```

3.1.3 Numerical Operators

VHDL contains the following numerical operators:

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
mod	Modulus
rem	Remainder
abs	Absolute value
**	Exponential

These operators work on types integer and real. Note that the default VHDL standard does not support numerical operators on types bit and bit_vector.

3.1.4 Relational Operators

VHDL contains the following relational operators. These operators compare two inputs of the same type and return the type Boolean (i.e., true or false).

Operator	Returns true if the comparison is:
=	Equal
/=	Not equal
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal

3.1.5 Shift Operators

VHDL contains the following shift operators. These operators work on vector types bit_vector and string.

Operator	Operation
sll	Shift left logical
srl	Shift right logical
sla	Shift left arithmetic
sra	Shift right arithmetic
rol	Rotate left
ror	Rotate right

The syntax for using a shift operation is to provide the name of the vector followed by the desired shift operator, followed by an integer indicating how many shift operations to perform. The target of the assignment must be the same type and size as the input.

Example:

```
A <= B srl 3;          -- A is assigned the result of a logical shift
                       -- right 3 times on B.
```

3.1.6 Concatenation Operator

In VHDL the **&** is used to concatenate multiple signals. The target of this operation must be the same size of the sum of the sizes of the input arguments.

Example:

```
Bus1 <= "11" & "00";        -- Bus1 must be 4-bits and will be assigned
                            -- the value "1100"

Bus2 <= BusA & BusB;        -- If BusA and BusB are 4-bits, then Bus2
                            -- must be 8-bits.

Bus3 <= '0' & BusA;         -- This attaches a leading '0' to BusA. Bus3
                            -- must be 5-bits
```

CONCEPT CHECK

CC3.1 Do all of the operators provided in the standard package work for all data types provided in the same package?

(A) Yes. Since both the operators and data types are in the same package, they all work together.

(B) No. Each operator only works on specific data types. It is up to the designer to know what types the operator work with.

3.2 Concurrent Signal Assignments with Logical Operators

Concurrent signal assignments are accomplished by simply using the <= operator after the begin statement in the architecture. Each individual assignment will be executed concurrently and synthesized as separate logic circuits. Consider the following example:

Example:

```
X <= A;
Y <= B;
Z <= C;
```

When simulated, these three lines of VHDL will make three separate signal assignments at the exact same time. This is different from a programming language that will first assign A to X, then B to Y, and finally C to Z. In VHDL this functionality is identical to three separate wires. This description will be directly synthesized into three separate wires.

Below is another example of how concurrent signal assignments in VHDL differ from a sequentially executed programming language:

Example:

```
A <= B;
B <= C;
```

In a VHDL simulation, the signal assignments of C to B and B to A will take place at the same time; however, during synthesis, the signal B will be eliminated from the design since this functionality

describes two wires in series. Automated synthesis tools will eliminate this unnecessary signal name. This is not the same functionality that would result if this example was implemented as a sequentially executed computer program. A computer program would execute the assignment of B to A first and then assign the value of C to B second. In this way, B represents a storage element that is passed to A before it is updated with C.

Each of the logical operators described in Sect. 3.1.2 can be used in conjunction with concurrent signal assignments to create individual combinational logic circuits.

3.2.1 Logical Operator Example: SOP Circuit

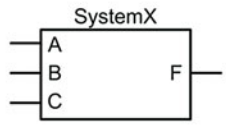
Example 3.1 shows how to design a VHDL model of a standard sum of products' combinational logic circuit using concurrent signal assignments with logical operators.

Example: Modeling Logic using Concurrent Signal Assignments and Logical Operators

Implement the following truth table using concurrent signal assignments with logical operators.

A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

First, let's design the entity. Let's call the entity *SystemX*. The entity will have three inputs (A, B, C) and one output (F). We'll use the type bit for all inputs/outputs so that this will synthesize directly into real circuitry.

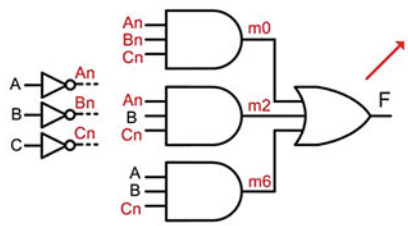



```
entity SystemX is
   port (A, B, C  :  in  bit;
         F        :  out bit);
end entity;
```

Now we design the architecture. We can create a canonical sum of products logic expression for this truth table using minterms.

$$F = \sum_{A,B,C}(0,2,6) = A'\cdot B'\cdot C' + A'\cdot B\cdot C' + A\cdot B\cdot C'$$

Drawing out the logic diagram will help us understand which internal signals need to be declared for the interim connections. Since there is a need for the complement of each of the inputs, the first set of logic will be three inverters. We'll need to create three signals to hold the inverted versions of the inputs. Let's call them An, Bn and Cn. We'll also need three signal to hold the outputs of the AND gates. Let's call them m0, m2 and m6. Using these internal signals, the port names from the entity, and logical operators, we can describe the behavior of the logic expression above.




```
architecture SystemX_arch of SystemX is

  signal  An, Bn, Cn : bit;
  signal  m0, m2, m6 : bit;

begin

  An <= not A;              -- NOT's
  Bn <= not B;
  Cn <= not C;

  m0 <= An and Bn and Cn; -- AND's
  m2 <= An and B  and Cn;
  m6 <= A  and B  and Cn;

  F  <= m0 or m2 or m6;    -- OR

end architecture;
```

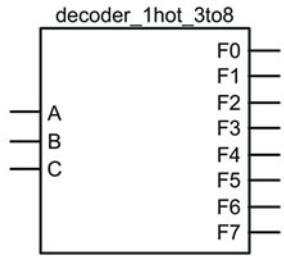
Example 3.1
SOP logic circuit: VHDL modeling using logical operators

3.2.2 Logical Operator Example: One-Hot Decoder

A one-hot decoder is a circuit that has *n* inputs and 2^n outputs. Each output will assert for one and only one input code. Since there are 2^n outputs, there will always be one and only one output asserted at any given time. Example 3.2 shows how to model a 3-to-8 one-hot decoder in VHDL with concurrent signal assignments and logic operators.

Example: 3-to-8 One-Hot Decoder – VHDL Modeling using Logical Operators

The block diagram and truth table for this system are as follows:



A	B	C	F7	F6	F5	F4	F3	F2	F1	F0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

To implement this in VHDL using logical operators, we must first determine the logic that will be used in the concurrent signal assignments. Again, since each logic function only has one input code corresponding to an output of '1', the minterm can be used to implement the logic.

$F0 = \sum_{A,B,C}(0) = A'\cdot B'\cdot C'$ $\quad F4 = \sum_{A,B,C}(4) = A\cdot B'\cdot C'$

$F1 = \sum_{A,B,C}(1) = A'\cdot B'\cdot C$ $\quad F5 = \sum_{A,B,C}(5) = A\cdot B'\cdot C$

$F2 = \sum_{A,B,C}(2) = A'\cdot B\cdot C'$ $\quad F6 = \sum_{A,B,C}(6) = A\cdot B\cdot C'$

$F3 = \sum_{A,B,C}(3) = A'\cdot B\cdot C$ $\quad F7 = \sum_{A,B,C}(7) = A\cdot B\cdot C$

In VHDL, each of the outputs requires a separate signal assignment.

```
entity decoder_1hot_3to8 is
   port (A,B,C                    : in  bit;
         F0,F1,F2,F3,F4,F5,F6,F7  : out bit);
end entity;

architecture decoder_1hot_3to8_arch of decoder_1hot_3to8 is

 begin

   F0 <= (not A) and (not B) and (not C);
   F1 <= (not A) and (not B) and (C);
   F2 <= (not A) and (B)     and (not C);
   F3 <= (not A) and (B)     and (C);
   F4 <= (A)     and (not B) and (not C);
   F5 <= (A)     and (not B) and (C);
   F6 <= (A)     and (B)     and (not C);
   F7 <= (A)     and (B)     and (C);

end architecture;
```

Example 3.2
3-to-8 One-hot decoder: VHDL modeling using logical operators

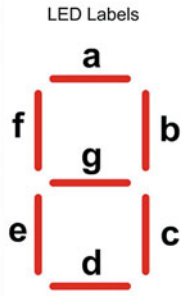
3.2.3 Logical Operator Example: 7-Segment Display Decoder

A 7-segment display decoder is a circuit used to drive character displays that are commonly found in applications such as digital clocks and household appliances. A character display is made up of seven individual LEDs, typically labeled a–g. The input to the decoder is the binary equivalent of the decimal or Hex character that is to be displayed. The output of the decoder is the arrangement of LEDs that will form the character. Decoders with 2-inputs can drive characters "0" to "3." Decoders with 3-inputs can drive characters "0" to "7." Decoders with 4-inputs can drive characters "0" to "F" with the case of the Hex characters being "A, b, c or C, d, E, and F."

Let's look at an example of how to design a 3-input, 7-segment decoder by hand. The first step in the process is to create the truth table for the outputs that will drive the LEDs in the display. We'll call these outputs F_a, F_b, . . ., F_g. Example 3.3 shows how to construct the truth table for the 7-segment display decoder. In this table, a logic 1 corresponds to the LED being ON.

Example: 7-Segment Display Decoder - Truth Table

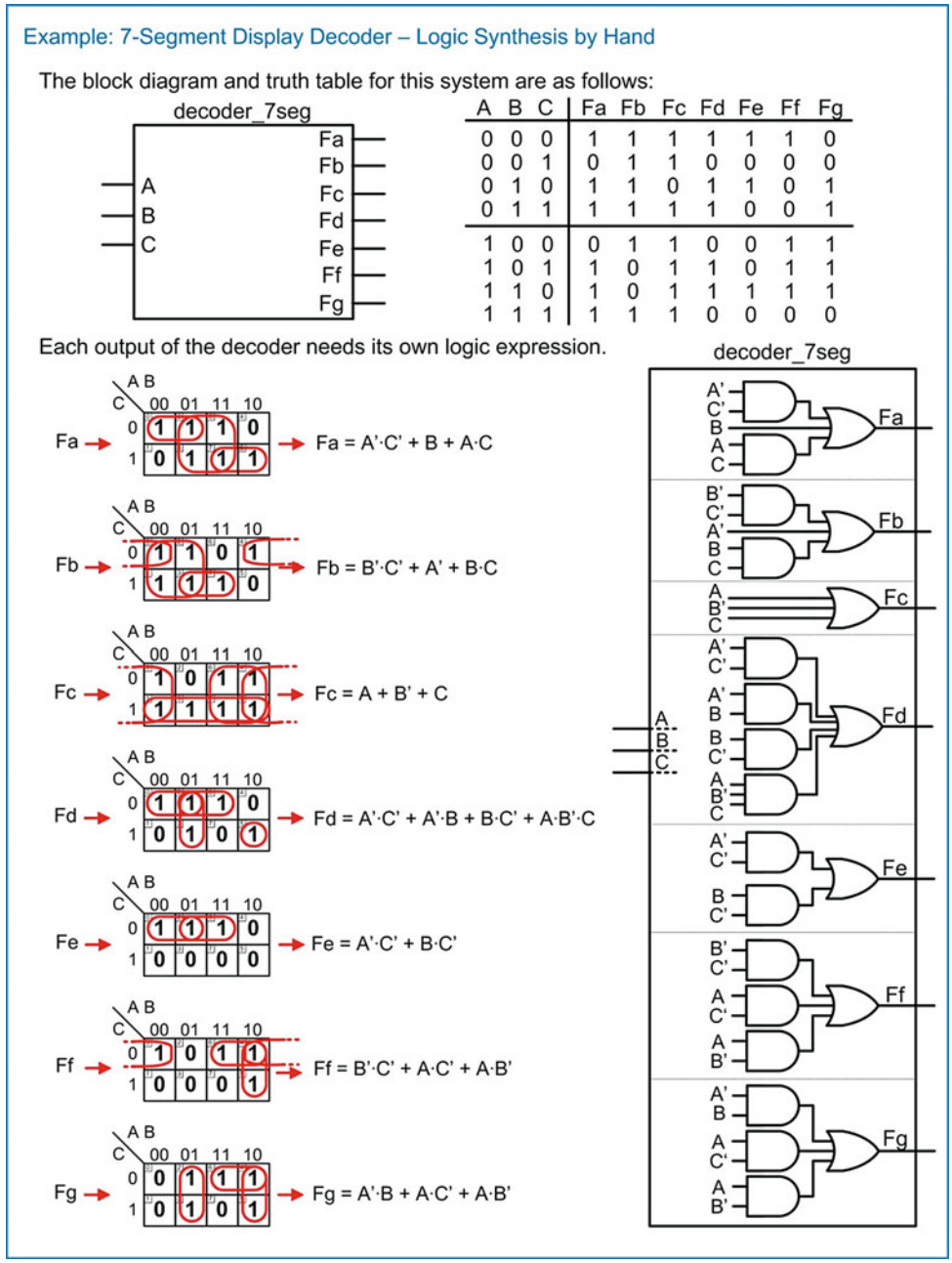
LED Labels



A	B	C		F_a	F_b	F_c	F_d	F_e	F_f	F_g
0	0	0		1	1	1	1	1	1	0
0	0	1		0	1	1	0	0	0	0
0	1	0		1	1	0	1	1	0	1
0	1	1		1	1	1	1	0	0	1
1	0	0		0	1	1	0	0	1	1
1	0	1		1	0	1	1	0	1	1
1	1	0		1	0	1	1	1	1	1
1	1	1		1	1	1	0	0	0	0

Example 3.3
7-Segment display decoder: truth table

If we wish to design this decoder by hand, we need to create seven separate combinational logic circuits. Each of the outputs (F_a–F_g) can be put into a 3-input K-map to find the minimized logic expression. Example 3.4 shows the design of the decoder from the truth table in Example 3.3 by hand.

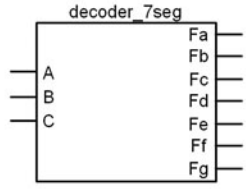


Example 3.4
7-Segment display decoder: logic synthesis by hand

This same functionality can be modeled in VHDL using concurrent signal assignments with logical operators. Example 3.5 shows how to model the 7-segment decoder in VHDL using concurrent signal assignments with logic operators. It should be noted that this example is somewhat artificial because a design would typically not be minimized before modeling in VHDL. Instead, model would be entered at the behavioral level, and then the CAD tool would be allowed to synthesize and minimize the final logic.

Example: 7-Segment Display Decoder – VHDL Modeling using Logical Operators

The block diagram and truth table for this system are as follows:



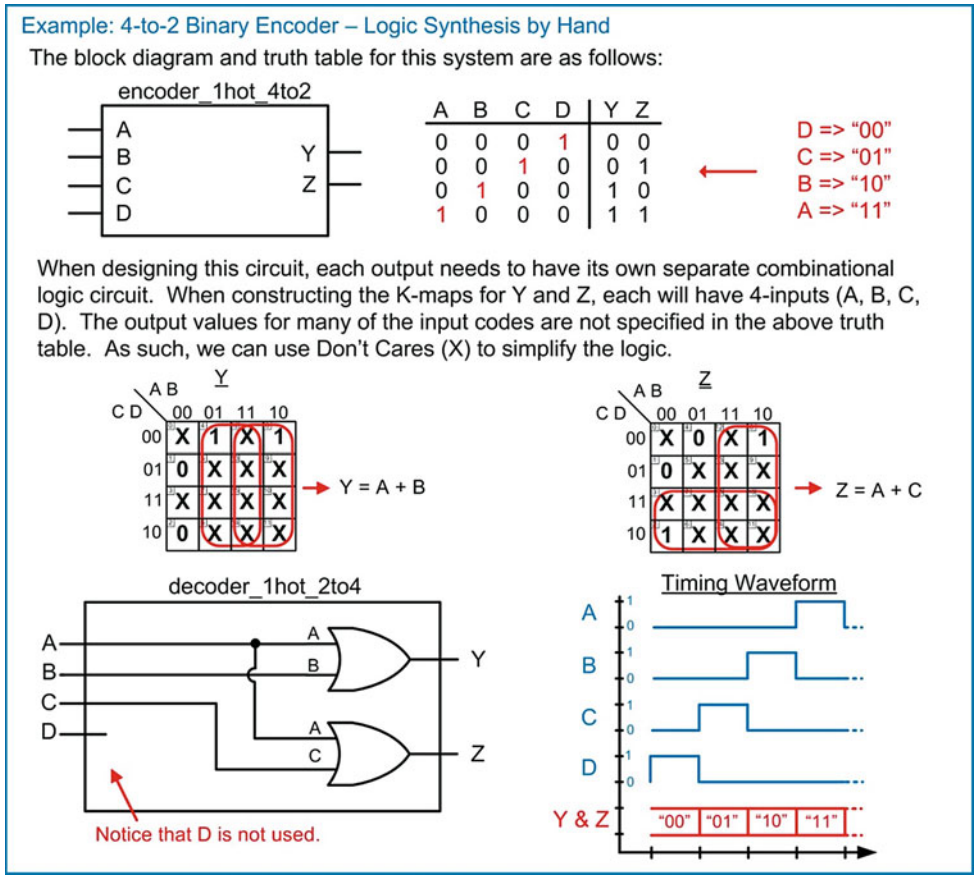
A	B	C	Fa	Fb	Fc	Fd	Fe	Ff	Fg
0	0	0	1	1	1	1	1	1	0
0	0	1	0	1	1	0	0	0	0
0	1	0	1	1	0	1	1	0	1
0	1	1	1	1	1	1	0	0	1
1	0	0	0	1	1	0	0	1	1
1	0	1	1	0	1	1	0	1	1
1	1	0	1	0	1	1	1	1	1
1	1	1	1	1	1	0	0	0	0

```
entity decoder_7seg is
   port (A,B,C                 : in  bit;
         Fa,Fb,Fc,Fd,Fe,Ff,Fg : out bit);
end entity;

architecture decoder_7seg_arch of decoder_7seg is

 begin

    Fa <= ((not A) and (not C)) or B or (A and C);
    Fb <= ((not B) and (not C)) or (not A) or (B and C);
    Fc <= A or (not B) or C;
    Fd <= ((not A) and (not C)) or ((not A) and B) or (B and (not C))
          or (A and (not B) and C);
    Fe <= ((not A) and (not C)) or (B and (not C));
    Ff <= ((not B) and (not C)) or (A and (not C)) or (A and (not B));
    Fg <= ((not A) and B) or (A and (not C)) or (A and (not B));

end architecture;
```

Example 3.5
7-Segment display decoder: VHDL modeling using logical operators

3.2.4 Logical Operator Example: One-Hot Encoder

A one-hot binary encoder has n outputs and 2^n inputs. The output will be an n-bit, binary code which corresponds to an assertion on one and only one of the inputs. Example 3.6 shows the process of designing a 4-to-2 binary encoder by hand (i.e., using the classical digital design approach).

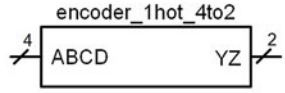


Example 3.6
4-to-2 Binary encoder: logic synthesis by hand

In VHDL this can be implemented directly using logical operators. Example 3.7 shows how to model the encoder in VHDL using concurrent signal assignments with logical operators.

Example: 4-to-2 Binary Encoder – VHDL Modeling using Logical Operators

The block diagram and truth table for this system are as follows:



ABCD	YZ
"0001"	"00"
"0010"	"01"
"0100"	"10"
"1000"	"11"

The following is the entity for this design that uses bit_vectors for the inputs and outputs.

```
entity encoder_1hot_4to2 is
    port (ABCD  : in  bit_vector(3 downto 0);
          YZ    : out bit_vector(1 downto 0));
end entity;
```

The following is an implementation of the encoder using concurrent signal assignments with logical operators.

```
architecture encoder_1hot_4to2_arch of encoder_1hot_4to2 is

 begin

  YZ(1) <= ABCD(3) or ABCD(2);
  YZ(0) <= ABCD(3) or ABCD(1);

end architecture;
```

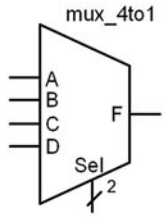
Example 3.7
4-to-2 Binary encoder: VHDL modeling using logical operators

3.2.5 Logical Operator Example: Multiplexer

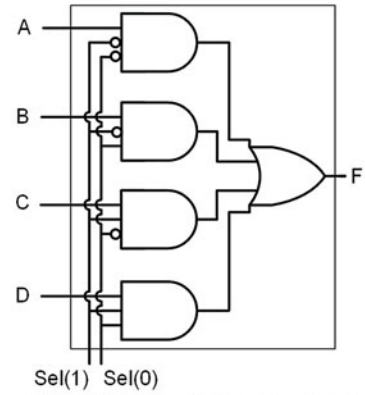
A multiplexer is a circuit that passes one of its multiple inputs to a single output based on a select input. This can be thought of as a digital routing switch. The multiplexer has n select lines, 2^n inputs, and one output. Example 3.8 shows the process of designing a 4-to-1 multiplexer using concurrent signal assignments and logical operators.

Example: 4-to-1 Multiplexer – VHDL Modeling using Logical Operators

The symbol and truth table for a 4-to-1 multiplexer are shown below. This can be implemented using a simple sum of products form based on the identity theorem and the appropriate inversions of the select line.



Sel	F
"00"	A
"01"	B
"10"	C
"11"	D



The following is the entity for this design that uses type bit_vector for the select input.

```
entity mux_4to1 is
    port (A,B,C,D : in  bit;
          Sel     : in  bit_vector(1 downto 0);
          F       : out bit);
end entity;
```

The following shows how to model the behavior of the mux using concurrent signal assignments with logical operators.

```
architecture mux_4to1_arch of mux_4to1 is
 begin
  F <= (A and not Sel(0) and not Sel(1)) or
       (B and not Sel(0) and     Sel(1)) or
       (C and     Sel(0) and not Sel(1)) or
       (D and     Sel(0) and     Sel(1));
end architecture;
```

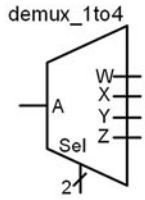
Example 3.8
4-to-1 Multiplexer: VHDL modeling using logical operators

3.2.6 Logical Operator Example: Demultiplexer

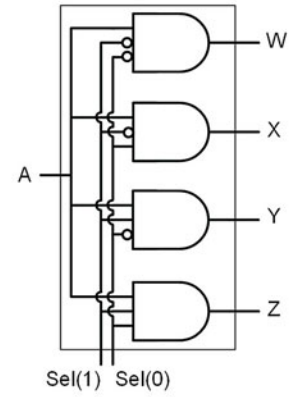
A demultiplexer works in a complementary fashion to a multiplexer. A demultiplexer has one input that is routed to one of its multiple outputs. The output that is active is dictated by a select input. A demux has n select lines that choose to route the input to one of its 2^n outputs. When an output is not selected, it outputs a logic 0. Example 3.9 shows the process of designing a 1-to-4 demultiplexer using concurrent signal assignments and logical operators.

Example: 1-to-4 Demultiplexer – VHDL Modeling using Logical Operators

The symbol and truth table for the 1-to-4 demultiplexer are shown below. This can be implemented using set of simple product terms based on the identity theorem and the appropriate inversions of the select line.



Sel	W	X	Y	Z
"00"	A	0	0	0
"01"	0	A	0	0
"10"	0	0	A	0
"11"	0	0	0	A



The following is the entity for this design that uses type bit_vector for the select input.

```
entity demux_1to4 is
    port (A       : in  bit;
          Sel     : in  bit_vector(1 downto 0);
          W,X,Y,Z : out bit);
end entity;
```

The following shows the behavior of the demux using concurrent signal assignments with logical operators.

```
architecture demux_1to4_arch of demux_1to4 is

 begin

  W <=  A and not Sel(0) and not Sel(1);
  X <=  A and not Sel(0) and     Sel(1);
  Y <=  A and     Sel(0) and not Sel(1);
  Z <=  A and     Sel(0) and     Sel(1);

end architecture;
```

Example 3.9
1-to-4 Demultiplexer: VHDL modeling using logical operators

Concept Check

CC3.2 Why does modeling combinational logic in its canonical form with concurrent signal assignments with logical operators defeat the purpose of the modern digital design flow?

(A) It requires the designer to first create the circuit using the classical digital design approach and then enter it into the HDL in a form that is essentially a text-based netlist. This doesn't take advantage of the abstraction capabilities and automated synthesis in the modern flow.

(B) It cannot be synthesized because the order of precedence of the logical operators in VHDL doesn't match the precedence defined in Boolean algebra.

(C) The circuit is in its simplest form so there is no work for the synthesizer to do.

(D) It doesn't allow an *else* clause to cover the outputs for any remaining input codes not explicitly listed.

3.3 Conditional Signal Assignments

Logical operators are good for describing the behavior of small circuits; however, in the prior example, we still needed to create the canonical or minimal sum of products logic expression by hand before describing the functionality in VHDL. The true power of an HDL is when the behavior of the system can be described fully without requiring any hand design. A conditional signal assignment allows us to describe a concurrent signal assignment using Boolean conditions that effect the values of the result. In a conditional signal assignment, the keyword **when** is used to describe the signal assignment for a particular Boolean condition. The keyword **else** is used to describe the signal assignments for any other conditions. Multiple Boolean conditions can be used to fully describe the output of the circuit under all input conditions. Logical operators can also be used in the Boolean conditions to create more sophisticated conditions. The Boolean conditions can be encompassed within parentheses for readability. The syntax for a conditional signal assignment is shown below.

```
signal_name <= expression_1 when condition_1 else
               expression_2 when condition_2 else
                            :
               expression_n;
```

Example:

```
F1 <= '0' when A='0' else '1';
F2 <= '1' when (A='0' and B='1') else '0';
F3 <= A when (C = D) else B;
```

An important consideration of conditional signal assignments is that they are still executed concurrently. Each assignment represents a separate, combinational logic circuit. In the above example, F1, F2, and F3 will be implemented as three separate, parallel circuits.

3.3.1 Conditional Signal Assignment Example: SOP Circuit

Example 3.10 shows how to design a VHDL model of a combinational logic circuit using conditional signal assignments. Note that this example uses the same truth table as in Example 3.1 to illustrate a comparison between approaches. This approach provides a model that can be created directly from the truth table without needing to do any synthesis or minimization by hand.

Example: Modeling Logic using Conditional Signal Assignments

Implement the following truth table using a conditional signal assignment.

A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

```
entity SystemX is
   port (A, B, C  :  in  bit;
         F        :  out bit);
end entity;
```

We can implement the entire truth table in its current form using a conditional signal assignment. While this is a verbose approach, it is sometimes more readable.

```
architecture SystemX_arch of SystemX is

begin

  F <= '1' when (A='0' and B='0' and C='0') else
       '0' when (A='0' and B='0' and C='1') else
       '1' when (A='0' and B='1' and C='0') else
       '0' when (A='0' and B='1' and C='1') else
       '0' when (A='1' and B='0' and C='0') else
       '0' when (A='1' and B='0' and C='1') else
       '1' when (A='1' and B='1' and C='0') else
       '0' when (A='1' and B='1' and C='1');

end architecture;
```

We can also reduce this into a more compressed form by only stating the input conditions that correspond to an output of '1' and using the "else" statement to produce an output of '0' for all other input codes.

```
architecture SystemX_arch of SystemX is

begin

  F <= '1' when (A='0' and B='0' and C='0') else
       '1' when (A='0' and B='1' and C='0') else
       '1' when (A='1' and B='1' and C='0') else
       '0';

end architecture;
```

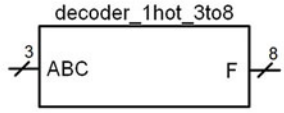
Example 3.10
SOP logic circuit: VHDL modeling using conditional signal assignments

3.3.2 Conditional Signal Assignment Example: One-Hot Decoder

Example 3.11 shows how to model a 3-to-8 one-hot decoder in VHDL with conditional signal assignments. Again, this approach allows the logic to be modeled directly from its functional description rather than having to perform any synthesis by hand.

Example: 3-to-8 One-Hot Decoder – VHDL Modeling – Conditional Signal Assignments

The block diagram and truth table for this system are as follows. Notice that the input and output ports now use type bit_vector in order to create a more compact description.



ABC	F(7)	F(6)	F(5)	F(4)	F(3)	F(2)	F(1)	F(0)
"000"	0	0	0	0	0	0	0	1
"001"	0	0	0	0	0	0	1	0
"010"	0	0	0	0	0	1	0	0
"011"	0	0	0	0	1	0	0	0
"100"	0	0	0	1	0	0	0	0
"101"	0	0	1	0	0	0	0	0
"110"	0	1	0	0	0	0	0	0
"111"	1	0	0	0	0	0	0	0

The following is the entity for this design that uses bit_vectors for the inputs and outputs.

```
entity decoder_1hot_3to8 is
    port (ABC  : in  bit_vector(2 downto 0);
          F    : out bit_vector(7 downto 0));
end entity;
```

The following is an implementation of the decoder using a conditional signal assignment. In this technique, the signal assignment can be made to the entire F vector instead of to the individual scalar outputs. This creates a compact VHDL model that will synthesis into eight separate combinational logic circuits.

```
architecture decoder_1hot_3to8_arch of decoder_1hot_3to8 is

 begin

  F <= "00000001" when (ABC = "000") else
       "00000010" when (ABC = "001") else
       "00000100" when (ABC = "010") else
       "00001000" when (ABC = "011") else
       "00010000" when (ABC = "100") else
       "00100000" when (ABC = "101") else
       "01000000" when (ABC = "110") else
       "10000000" when (ABC = "111");

end architecture;
```

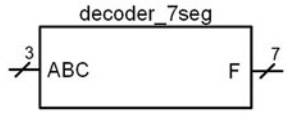
Example 3.11
3-to-8 One-hot decoder: VHDL modeling using conditional signal assignments

3.3.3 Conditional Signal Assignment Example: 7-Segment Display Decoder

Back in Example 3.3 the truth table for a 7-segment display decoder was given along with the subsequent steps to create its logic using the classical digital design approach and model it in VHDL using logical operators. With a conditional signal assignment, this decoder can be modeled directly from the truth table without needing to do any design by hand. Example 3.12 shows how to model the logic for a 7-segment display decoder using a conditional signal assignment.

Example: 7-Segment Decoder – VHDL Modeling – Conditional Signal Assignments

The block diagram and truth table for this system are as follows:



	a	b	c	d	e	f	g
ABC	F(6)	F(5)	F(4)	F(3)	F(2)	F(1)	F(0)
"000"	1	1	1	1	1	1	0
"001"	0	1	1	0	0	0	0
"010"	1	1	0	1	1	0	1
"011"	1	1	1	1	0	0	1
"100"	0	1	1	0	0	1	1
"101"	1	0	1	1	0	1	1
"110"	1	0	1	1	1	1	1
"111"	1	1	1	0	0	0	0

The following is the entity for this design that uses bit_vectors for the inputs and outputs.

```
entity decoder_7seg is
  port (ABC  : in  bit_vector(2 downto 0);
        F    : out bit_vector(6 downto 0));
end entity;
```

The following shows a way to implement the behavior of the 7-segment display decoder using a conditional signal assignment.

```
architecture decoder_7seg_arch of decoder_7seg is
 begin
   F <= "1111110" when (ABC = "000") else
        "0110000" when (ABC = "001") else
        "1101101" when (ABC = "010") else
        "1111001" when (ABC = "011") else
        "0110011" when (ABC = "100") else
        "1011011" when (ABC = "101") else
        "1011111" when (ABC = "110") else
        "1110000" when (ABC = "111");
end architecture;
```

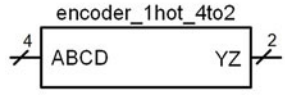
Example 3.12
7-Segment display decoder: VHDL modeling using conditional signal assignments

3.3.4 Conditional Signal Assignment Example: One-Hot Encoder

Example 3.13 shows how to model a one-hot encoder in VHDL with conditional signal assignments. Again, this approach allows the logic to be modeled directly from its functional description rather than having to perform any synthesis by hand.

Example: 4-to-2 Binary Encoder – VHDL Modeling using Conditional Signal Assignments

The block diagram and truth table for this system are as follows:



ABCD	YZ
"0001"	"00"
"0010"	"01"
"0100"	"10"
"1000"	"11"

The following is the entity for this design that uses bit_vectors for the inputs and outputs.

```
entity encoder_1hot_4to2 is
    port (ABCD  : in  bit_vector(3 downto 0);
          YZ    : out bit_vector(1 downto 0));
end entity;
```

The following is an implementation of the encoder using a conditional signal assignment.

```
architecture encoder_1hot_4to2_arch of encoder_1hot_4to2 is

 begin

  YZ <= "00" when (ABCD = "0001") else
        "01" when (ABCD = "0010") else
        "10" when (ABCD = "0100") else
        "11" when (ABCD = "1000") else
        "00";

end architecture;
```

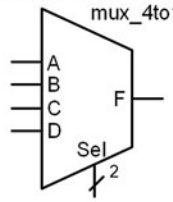
Example 3.13
4-to-2 Binary encoder: VHDL modeling using conditional signal assignments

3.3.5 Conditional Signal Assignment Example: Multiplexer

Example 3.14 shows the process of designing a 4-to-1 multiplexer using conditional signal assignments.

Example: 4-to-1 Multiplexer – VHDL Modeling using Conditional Signal Assignments

The symbol and truth table for a 4-to-1 multiplexer are shown below. This behavior can be implemented directly with a conditional signal assignment.



Sel	F
"00"	A
"01"	B
"10"	C
"11"	D

The following is the entity for this design that uses type bit_vector for the select input.

```
entity mux_4to1 is
    port (A,B,C,D : in  bit;
          Sel     : in  bit_vector(1 downto 0);
          F       : out bit);
end entity;
```

The following shows how to model the behavior of the mux using a conditional signal assignment.

```
architecture mux_4to1_arch of mux_4to1 is
 begin
  F <= A when (Sel = "00") else
       B when (Sel = "01") else
       C when (Sel = "10") else
       D when (Sel = "11");
end architecture;
```

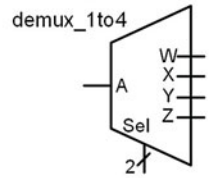
Example 3.14
4-to-1 Multiplexer: VHDL modeling using conditional signal assignments

3.3.6 Conditional Signal Assignment Example: Demultiplexer

Example 3.15 shows the process of designing a 1-to-4 demultiplexer using conditional signal assignments.

Example: 1-to-4 Demultiplexer – VHDL Modeling using Conditional Signal Assignments

The symbol and truth table for the 1-to-4 demultiplexer are shown below. This behavior can be implemented directly with conditional signal assignments.



Sel	W	X	Y	Z
"00"	A	0	0	0
"01"	0	A	0	0
"10"	0	0	A	0
"11"	0	0	0	A

The following is the entity for this design that uses type bit_vector for the select input.

```
entity demux_1to4 is
    port (A       : in  bit;
          Sel     : in  bit_vector(1 downto 0);
          W,X,Y,Z : out bit);
end entity;
```

The following shows the behavior of the demux using conditional signal assignments.

```
architecture demux_1to4_arch of demux_1to4 is

 begin

  W <= A when (Sel = "00") else '0';
  X <= A when (Sel = "01") else '0';
  Y <= A when (Sel = "10") else '0';
  Z <= A when (Sel = "11") else '0';

end architecture;
```

Example 3.15
1-to-4 Demultiplexer: VHDL modeling using conditional signal assignments

Concept Check

CC3.3 Why does a conditional signal assignment better reflect the modern digital design flow compared to a concurrent signal assignment with logical operators?

(A) A conditional signal assignment allows the logic to be modeled directly from its functional description as opposed to a logical operator approach where the logic expressions must be determined prior to HDL modeling. This allows the conditional signal assignment approach to take advantage of automated synthesis and avoids any hand design.

(B) A conditional signal assignment uses an "else" keyword, which makes it more like a programming language operator.

(C) The conditional signal assignment can model the entire logic circuit in one assignment while the logical operator approach often takes multiple separate assignments.

(D) The *else* clause allows coverage for outputs for any remaining input codes not explicitly listed.

3.4 Selected Signal Assignments

A selected signal assignment provides another technique to implement concurrent signal assignments. In this approach, the signal assignment is based on a specific value on the input signal. The keyword **with** is used to begin the selected signal assignment. It is then followed by the name of the input that will be used to dictate the value of the output. Only a single variable name can be listed as the input. This means that if the assignment is going to be based on multiple variables, they must first be concatenated into a single vector name before starting the selected signal assignment. After the input is listed, the keyword **select** signifies the beginning of the signal assignments. An assignment is made to a signal based on a list of possible input values that follow the keyword **when**. Multiple values of the input codes can be used and are separated by commas. The keyword **others** is used to cover any input values that are not explicitly stated. The syntax for a selected signal assignment is as follows:

```
with input_name select
     signal_name <= expression_1 when condition_1,
                    expression_2 when condition_2,
                                 :
                    expression_n when others;
```

Example:

```
with A select
  F1 <= '1' when '0',    -- F1 will be assigned '1' when A='0'
        '0' when '1';    -- F1 will be assigned '0' when A='1'

AB <= A&B;               -- concatenate A and B so that they can be used as a vector
with AB select
  F2 <= '0' when "00",   -- F2 will be assigned '0' when AB="00"
        '1' when "01",
        '1' when "10",
        '0' when "11";

with AB select
  F3 <= '1' when "01",
        '1' when "10",
        '0' when others;
```

One feature of selected signal assignments that makes its form even more compact than other techniques is that multiple input codes that correspond to the same output assignment can be listed on the same line pipe (|)-delimited. The example for F3 can be equivalently described as:

```
with AB select
  F3 <= '1' when "01" | "10",
        '0' when others;
```

3.4.1 Selected Signal Assignment Example: SOP Circuit

Example 3.16 shows how to design a VHDL model of a combinational logic circuit using selected signal assignments. Note that this example uses the same truth table as in Example 3.1 to illustrate a comparison between approaches. This approach provides a model that can be created directly from the truth table without needing to do any synthesis or minimization by hand.

Example: Modeling Logic using Selected Signal Assignments

Implement the following truth table using a selected signal assignment.

A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

```
entity SystemX is
   port (A, B, C  :  in  bit;
         F        :  out bit);
end entity;
```

We can implement the entire truth table in its current form using a selected signal assignment. Since we are basing our output values on three separate scalar inputs, we need to concatenate them into a vector so that the new vector name can be used as the input in the selected signal assignment. We'll first declare a new signal called "ABC" of type bit_vector(2 downto 0). After the begin statement, we'll assign the concatenation of A, B and C to this vector. The new vector name can now be used as an input.

```
architecture SystemX_arch of SystemX is

  signal ABC : bit_vector(2 downto 0);

begin

  ABC <= A & B & C;

  with (ABC) select
     F <= '1' when "000",
          '0' when "001",
          '1' when "010",
          '0' when "011",
          '0' when "100",
          '0' when "101",
          '1' when "110",
          '0' when "111";

end architecture;
```

We can reduce the size of the selected signal assignment by only listing the input codes corresponding to an output of '1' and use the "others" keyword to handle all input codes corresponding to an output of '0'.

```
architecture SystemX_arch of SystemX is

  signal ABC : bit_vector(2 downto 0);

begin

  ABC <= A & B & C;

  with (ABC) select
     F <= '1' when "000",
          '1' when "010",
          '1' when "110",
          '0' when others;

end architecture;
```

We can further reduce the size of the selected signal assignment by pipe delimiting the input codes corresponding to an output of '1'.

```
architecture SystemX_arch of SystemX is

  signal ABC : bit_vector(2 downto 0);

begin

  ABC <= A & B & C;

  with (ABC) select
     F <= '1' when "000"|"010"|"110",
          '0' when others;

end architecture;
```

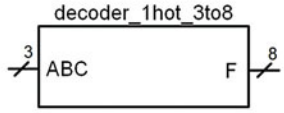
Example 3.16
SOP Logic circuit: VHDL modeling using selected signal assignments

3.4.2 Selected Signal Assignment Example: One-Hot Decoder

Example 3.17 shows how to model a 3-to-8 one-hot decoder in VHDL with selected signal assignments. Again, this approach allows the logic to be modeled directly from its functional description rather than having to perform any synthesis by hand.

Example: 3-to-8 One-Hot Decoder – VHDL Modeling – Selected Signal Assignments

The block diagram and truth table for this system are as follows. Notice that the input and output ports now use type bit_vector in order to create a more compact description.



ABC	F(7)	F(6)	F(5)	F(4)	F(3)	F(2)	F(1)	F(0)
"000"	0	0	0	0	0	0	0	1
"001"	0	0	0	0	0	0	1	0
"010"	0	0	0	0	0	1	0	0
"011"	0	0	0	0	1	0	0	0
"100"	0	0	0	1	0	0	0	0
"101"	0	0	1	0	0	0	0	0
"110"	0	1	0	0	0	0	0	0
"111"	1	0	0	0	0	0	0	0

The following is the entity for this design that uses bit_vectors for the inputs and outputs.

```
entity decoder_1hot_3to8 is
    port (ABC  : in  bit_vector(2 downto 0);
          F    : out bit_vector(7 downto 0));
end entity;
```

The following is an implementation of the decoder using a selected signal assignment. In this technique, the signal assignment can be made to the entire F vector instead of to the individual scalar outputs. This creates a compact VHDL model that will synthesis into eight separate combinational logic circuits.

```
architecture decoder_1hot_3to8_arch of decoder_1hot_3to8 is

 begin

  with (ABC) select
     F <= "00000001" when "000",
          "00000010" when "001",
          "00000100" when "010",
          "00001000" when "011",
          "00010000" when "100",
          "00100000" when "101",
          "01000000" when "110",
          "10000000" when "111";

end architecture;
```

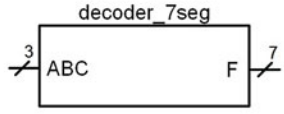
Example 3.17
3-to-8 One-hot decoder: VHDL modeling using selected signal assignments

3.4.3 Selected Signal Assignment Example: 7-Segment Display Decoder

Back in Example 3.3 the truth table for a 7-segment display decoder was given along with the subsequent steps to create its logic using the classical digital design approach and model it in VHDL using logical operators. With a selected signal assignment, this decoder can be modeled directly from the truth table without needing to do any design by hand. Example 3.18 shows how to model the logic for a 7-segment display decoder using a selected signal assignment.

Example: 7-Segment Decoder – VHDL Modeling – Selected Signal Assignments

The block diagram and truth table for this system are as follows:



	a	b	c	d	e	f	g
ABC	F(6)	F(5)	F(4)	F(3)	F(2)	F(1)	F(0)
"000"	1	1	1	1	1	1	0
"001"	0	1	1	0	0	0	0
"010"	1	1	0	1	1	0	1
"011"	1	1	1	1	0	0	1
"100"	0	1	1	0	0	1	1
"101"	1	0	1	1	0	1	1
"110"	1	0	1	1	1	1	1
"111"	1	1	1	0	0	0	0

The following is the entity for this design that uses bit_vectors for the inputs and outputs.

```
entity decoder_7seg is
  port (ABC  : in  bit_vector(2 downto 0);
        F    : out bit_vector(6 downto 0));
end entity;
```

The following shows a way to implement the behavior of the 7-segment display decoder using a selected signal assignment.

```
architecture decoder_7seg_arch of decoder_7seg is
 begin
   with (ABC) select
    F <= "1111110" when "000",
         "0110000" when "001",
         "1101101" when "010",
         "1111001" when "011",
         "0110011" when "100",
         "1011011" when "101",
         "1011111" when "110",
         "1110000" when "111";
end architecture;
```

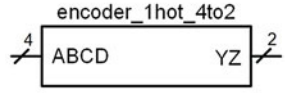
Example 3.18
7-Segment display decoder: VHDL modeling using selected signal assignments

3.4.4 Selected Signal Assignment Example: One-Hot Encoder

Example 3.19 shows how to model a one-hot encoder in VHDL with selected signal assignments. Again, this approach allows the logic to be modeled directly from its functional description rather than having to perform any synthesis by hand.

Example: 4-to-2 Binary Encoder – VHDL Modeling using Selected Signal Assignments

The block diagram and truth table for this system are as follows:



ABCD	YZ
"0001"	"00"
"0010"	"01"
"0100"	"10"
"1000"	"11"

The following is the entity for this design that uses bit_vectors for the inputs and outputs.

```
entity encoder_1hot_4to2 is
    port (ABCD  : in  bit_vector(3 downto 0);
          YZ    : out bit_vector(1 downto 0));
end entity;
```

The following is an implementation of the encoder using a selected signal assignment.

```
architecture encoder_1hot_4to2_arch of encoder_1hot_4to2 is

 begin

  with (ABCD) select
    YZ <= "00" when "0001",
          "01" when "0010",
          "10" when "0100",
          "11" when "1000",
          "00" when others;

end architecture;
```

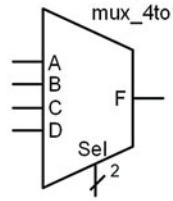
Example 3.19
4-to-2 Binary encoder: VHDL modeling using selected signal assignments

3.4.5 Selected Signal Assignment Example: Multiplexer

Example 3.20 shows the process of designing a 4-to-1 multiplexer using selected signal assignments.

Example: 4-to-1 Multiplexer – VHDL Modeling using Selected Signal Assignments

The symbol and truth table for a 4-to-1 multiplexer are shown below. This behavior can be implemented directly with a selected signal assignment.



Sel	F
"00"	A
"01"	B
"10"	C
"11"	D

The following is the entity for this design that uses type bit_vector for the select input.

```
entity mux_4to1 is
    port (A,B,C,D : in  bit;
          Sel     : in  bit_vector(1 downto 0);
          F       : out bit);
end entity;
```

The following shows how to model the behavior of the mux using a selected signal assignment.

```
architecture mux_4to1_arch of mux_4to1 is
 begin
   with (Sel) select
     F <= A when "00",
          B when "01",
          C when "10",
          D when "11";
end architecture;
```

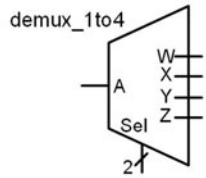
Example 3.20
4-to-1 Multiplexer: VHDL modeling using selected signal assignments

3.4.6 Selected Signal Assignment Example: Demultiplexer

Example 3.21 shows the process of designing a 1-to-4 demultiplexer using selected signal assignments.

Example: 1-to-4 Demultiplexer – VHDL Modeling using Selected Signal Assignments

The symbol and truth table for the 1-to-4 demultiplexer are shown below. This behavior can be implemented directly with selected signal assignments.



Sel	W	X	Y	Z
"00"	A	0	0	0
"01"	0	A	0	0
"10"	0	0	A	0
"11"	0	0	0	A

The following is the entity for this design that uses type bit_vector for the select input.

```
entity demux_1to4 is
    port (A       : in  bit;
          Sel     : in  bit_vector(1 downto 0);
          W,X,Y,Z : out bit);
end entity;
```

The following shows the behavior of the demux using selected signal assignments.

```
architecture demux_1to4_arch of demux_1to4 is
 begin
   with (Sel) select
     W <= A when "00",'0' when others;

   with (Sel) select
     X <= A when "01",'0' when others;

   with (Sel) select
     Y <= A when "10",'0' when others;

   with (Sel) select
     Z <= A when "11",'0' when others;
end architecture;
```

Example 3.21
1-to-4 Demultiplexer: VHDL modeling using selected signal assignments

Concept Check

CC3.4 Why does a selected signal assignment often require a separate concatenation operation?

(A) Concatenating the inputs makes the assignment easier to read.

(B) A selected signal assignment only support a single signal name for its input. If it is desired to look at multiple signal names, they must first be concatenated together to form a new signal name for use in the selected signal assignment.

(C) Since there is not an else clause, the selected signal assignment needs a way to handle the outputs for input codes not explicitly listed.

(D) To avoid having to use multiple parentheses in the input signal list.

3.5 Delayed Signal Assignments

3.5.1 Inertial Delay

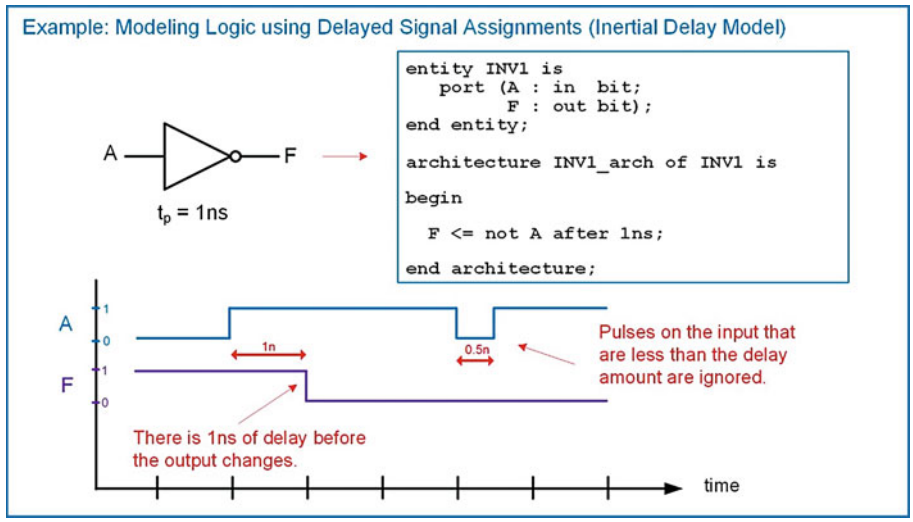
VHDL provides the ability to delay a concurrent signal assignment in order to more accurately model the behavior of real gates. The keyword **after** is used to delay an assignment by a certain amount of time. The magnitude of the delay is provided as type time. The syntax for delaying an assignment is as follows:

```
signal_name <= <expression> after <time>;
```

Example:

```
A <= B after 3us;
C <= D and E after 10ns;
```

If an input pulse is shorter in duration than the amount of the delay, the input pulse is ignored. This is called the *inertial delay model*. Example 3.22 shows how to design a VHDL model with a delayed signal assignment using the inertial delay model.



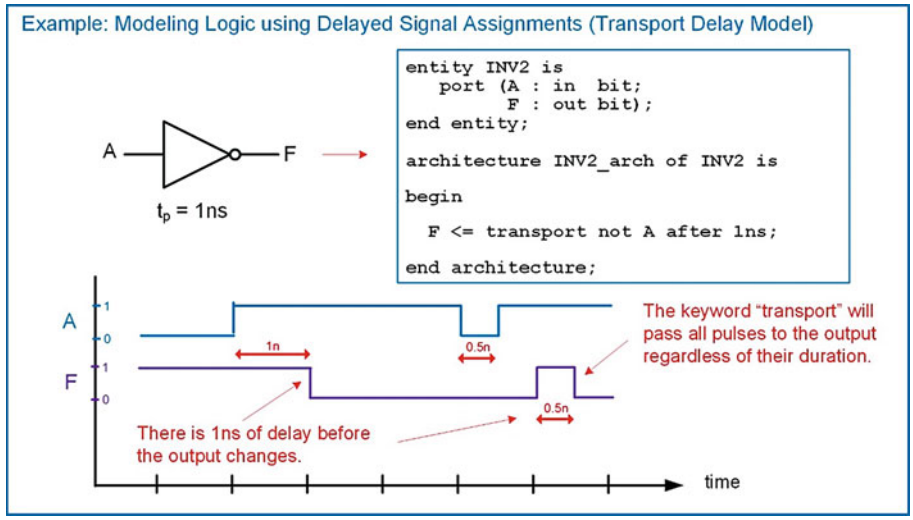
Example 3.22
Modeling logic using delayed signal assignments (inertial delay model)

3.5.2 Transport Delay

Ignoring brief input pulses on the input accurately models the behavior of on-chip gates. When the input pulse is faster than the delay of the gate, the output of the gate does not have time to respond. As a result, there will not be a logic change on the output. If it is desired to have all pulses on the inputs show up on the outputs when modeling the behavior of other types of digital logic, the keyword **transport** is used in conjunction with the after statement. This is called the *transport delay model*.

```
signal_name <= transport <expression> after <time>;
```

Example 3.23 shows how to perform a delayed signal assignment using the transport delay model.



Example 3.23
Modeling logic using delayed signal assignments (transport delay model)

Concept Check

CC3.5 Can a delayed signal assignment impact multiple concurrent signal assignments?

(A) Yes. If a signal assignment with delay is made to a signal that is also used as an input in a separate concurrent signal assignment, then the delay will propagate through both assignments.

(B) No. Only the assignment in which the delay is used will experience the delay.

Summary

- VHDL operators are defined to work on specific data types. Not all operators work on all types within a package.
- *Concurrency* is the term that describes operations being performed in parallel. This allows real-world system behavior to be modeled.
- VHDL contains three direct techniques to model concurrent logic behavior. These are *concurrent signal assignments with logical operators*, *conditional signal assignments*, and *selected signal assignments*.
- Delay can be modeled in VHDL using either the *inertial* or *transport* model. Inertial delay will ignore pulses that are shorter than the delay amount, while transport delay will pass all transitions.

Exercise Problems

Section 3.1: VHDL Operators

3.1.1 What data types do the logical operators in the standard package work on?

3.1.2 Which logical operator has the highest priority when evaluating the order of precedence of operations?

3.1.3 If parentheses are not used in a signal assignment with logical operators, how is the order of precedence determined?

3.1.4 What data types do the numerical operators in the standard package work on?

3.1.5 What is the return type of a relational operator?

Section 3.2: Concurrent Signal Assignments with Logical Operators

3.2.1 Design a VHDL model to implement the behavior described by the 3-input minterm list shown in Fig. 3.1. Use concurrent signal assignments and logical operators. Declare your entity to match the block diagram provided. Use the type bit for your ports.

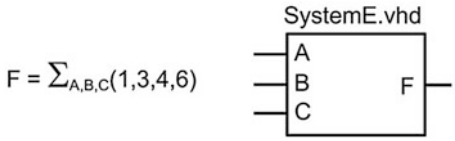


Fig. 3.1
System E functionality

3.2.2 Design a VHDL model to implement the behavior described by the 3-input maxterm list shown in Fig. 3.2. Use concurrent signal assignments and logical operators. Declare your entity to match the block diagram provided. Use the type bit for your ports.

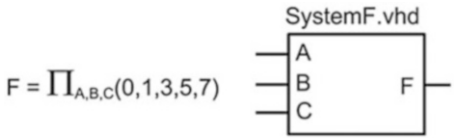


Fig. 3.2
System F functionality

3.2.3 Design a VHDL model to implement the behavior described by the 3-input truth table shown in Fig. 3.3. Use concurrent signal assignments and logical operators. Declare your entity to match the block diagram provided. Use the type bit for your ports.

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

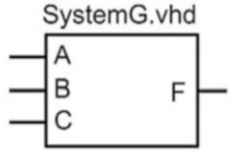


Fig. 3.3
System G functionality

3.2.4 Design a VHDL model to implement the behavior described by the 4-input minterm list shown in Fig. 3.4. Use concurrent signal assignments and logical operators. Declare your entity to match the block diagram provided. Use the type bit for your ports.

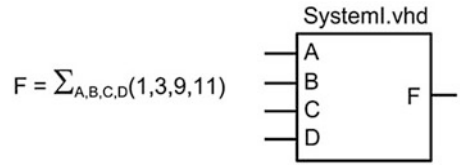


Fig. 3.4
System I functionality

3.2.5 Design a VHDL model to implement the behavior described by the 4-input maxterm list shown in Fig. 3.5. Use concurrent signal assignments and logical operators. Declare your entity to match the block diagram provided. Use the type bit for your ports.

Fig. 3.5
System J functionality

3.2.6 Design a VHDL model to implement the behavior described by the 4-input truth table shown in Fig. 3.6. Use concurrent signal assignments and logical operators. Declare your entity to match the block diagram provided. Use the type bit for your ports.

A	B	C	D	F
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

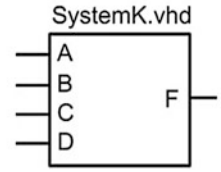


Fig. 3.6
System K functionality

Section 3.3: Conditional Signal Assignments

3.3.1 Design a VHDL model to implement the behavior described by the 3-input minterm list shown in Fig. 3.1. Use conditional signal assignments. Declare your entity to match the block diagram provided. Use the type bit for your ports.

3.3.2 Design a VHDL model to implement the behavior described by the 3-input maxterm list shown in Fig. 3.2. Use conditional signal assignments. Declare your entity to match the block diagram provided. Use the type bit for your ports.

3.3.3 Design a VHDL model to implement the behavior described by the 3-input truth table shown in Fig. 3.3. Use conditional signal assignments. Declare your entity to match the block diagram provided. Use the type bit for your ports.

3.3.4 Design a VHDL model to implement the behavior described by the 4-input minterm list shown in Fig. 3.4. Use conditional signal assignments. Declare your entity to match the block diagram provided. Use the type bit for your ports.

3.3.5 Design a VHDL model to implement the behavior described by the 4-input maxterm list shown in Fig. 3.5. Use conditional signal assignments. Declare your entity to match the block diagram provided. Use the type bit for your ports.

3.3.6 Design a VHDL model to implement the behavior described by the 4-input truth table shown in Fig. 3.6. Use conditional signal assignments. Declare your entity to match the block diagram provided. Use the type bit for your ports.

Section 3.4: Selected Signal Assignments

3.4.1 Design a VHDL model to implement the behavior described by the 3-input minterm list shown in Fig. 3.1. Use selected signal assignments. Declare your entity to match the block diagram provided. Use the type bit for your ports.

3.4.2 Design a VHDL model to implement the behavior described by the 3-input maxterm list shown in Fig. 3.2. Use selected signal assignments. Declare your entity to match the block diagram provided. Use the type bit for your ports.

3.4.3 Design a VHDL model to implement the behavior described by the 3-input truth table shown in Fig. 3.3. Use selected signal assignments. Declare your entity to match the block diagram provided. Use the type bit for your ports.

3.4.4 Design a VHDL model to implement the behavior described by the 4-input minterm list shown in Fig. 3.4. Use selected signal assignments. Declare your entity to match the block diagram provided. Use the type bit for your ports.

3.4.5 Design a VHDL model to implement the behavior described by the 4-input maxterm list shown in Fig. 3.5. Use selected signal assignments. Declare your entity to match the block diagram provided. Use the type bit for your ports.

3.4.6 Design a VHDL model to implement the behavior described by the 4-input truth table shown in Fig. 3.6. Use selected signal assignments. Declare your entity to match the block diagram provided. Use the type bit for your ports.

Section 3.5: Delayed Signal Assignments

3.5.1 Design a VHDL model to implement the behavior described by the 3-input minterm list shown in Fig. 3.1. Use concurrent signal assignments and logical operators. Create the model so that every logic operation has 1 ns of inertial delay. Declare your entity to match the block diagram provided. Use the type bit for your ports.

3.5.2 Design a VHDL model to implement the behavior described by the 3-input maxterm list shown in Fig. 3.2. Use concurrent signal assignments and logical operators. Create the model so that every logic operation has 1 ns of inertial delay. Declare your entity to match the block diagram provided. Use the type bit for your ports.

3.5.3 Design a VHDL model to implement the behavior described by the 3-input truth table shown in Fig. 3.3. Use concurrent signal assignments and logical operators. Create the model so that every logic operation has 1 ns of inertial delay. Declare your entity to match the block diagram provided. Use the type bit for your ports.

3.5.4 Design a VHDL model to implement the behavior described by the 4-input minterm list shown in Fig. 3.4. Use concurrent signal assignments and logical operators. Create the model so that every logic operation has 1 ns of transport delay. Declare your entity to match the block diagram provided. Use the type bit for your ports.

3.5.5 Design a VHDL model to implement the behavior described by the 4-input maxterm list shown in Fig. 3.5. Use concurrent signal assignments and logical operators. Create the model so that every logic operation has 1 ns of transport delay. Declare your entity to match the block diagram provided. Use the type bit for your ports.

3.5.6 Design a VHDL model to implement the behavior described by the 4-input truth table shown in Fig. 3.6. Use concurrent signal assignments and logical operators. Create the model so that every logic operation has 1 ns of transport delay. Declare your entity to match the block diagram provided. Use the type bit for your ports.