Combinational Circuit Design using VHDL

Concurrent Assignment Statements

The concurrent assignment statements that are only used in combinational circuit descriptions are the simple signal assignment, the selected signal assignment and the conditional signal assignment.

Simple Signal Assignment

Figure 1 shows a 1-bit full adder (`fadder`).

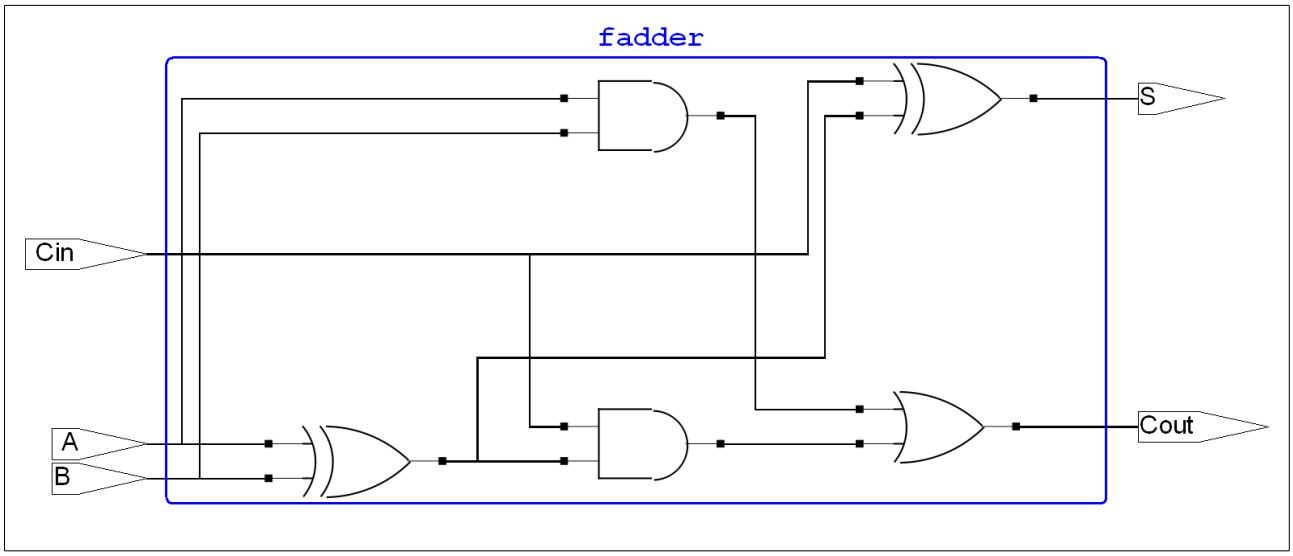

Figure 1

The external – outside of the blue rectangle – description of `fadder` is an input/output description (ports `A`, `B` and `Cin` are the circuit's inputs and ports `S` and `Cout` are the circuit's outputs). The internal – inside of the blue rectangle – description of `fadder` is a functionality description. The VHDL description (VHDL code) for the `fadder` circuit is shown in **Figure 2**.

```
 1  -- 1-bit FULL ADDER.
 2
 3  library IEEE;
 4  use IEEE.STD_LOGIC_1164.ALL;
 5
 6  entity fadder is
 7     Port(A,B,Cin : in  STD_LOGIC;
 8          S,Cout  : out  STD_LOGIC);
 9  end fadder;
10
11  architecture Behavioral of fadder is
12  begin
13     S    <= A xor B xor Cin; -- Simple Assignment Statement.
14     Cout <= (A and B) or ((A xor B) and Cin);
15  end Behavioral;
```

Figure 2

The digital circuit designer writes VHDL code in a text editor and saves the file with a **.vhd** extension (the code shown in **Figure 2** could have been saved in a file named **fadder.vhd**). Line numbers shown in **Figure 2** (and in figures that follow) are not part of the VHDL code. They are used only for reference.

Line `1` represents a comment in the code. Text in a specific line that follows the "`--`" character combination is considered to be a comment (see also line `13`).

A library is a collection of packages and every package consists of one or more VHDL source files. Line `3` in **Figure 2** states that packages from the `IEEE` library will be used in `fadder`'s source code and line `4` states that all the parts of the `STD_LOGIC_1164` package will be used in `fadder`'s source code.

The `STD_LOGIC_1164` package defines the `STD_LOGIC` type. The following values are legal for a `STD_LOGIC` data object : 0, 1, Z, –, L, H, U, X, and W. Only the first four are useful for synthesis of logic circuits. The value Z represents high impedance, and – stands for "don't care." The value L stands for "weak 0," H means "weak 1," U means "uninitialized," X means "unknown," and W means "weak unknown".

Lines `6-9` in **Figure 2** represent the entity declaration. The entity declaration is the external description of the `fadder` circuit. It defines the ports of `fadder` and the mode of each port. **Figure 3** summarizes the available port modes.

Mode	Purpose
IN	Used for a signal that is an input to an entity.
OUT	Used for a signal that is an output from an entity. The value of the signal can not be used inside the entity. This means that in an assignment statement, the signal can appear only to the left of the <= operator.
INOUT	Used for a signal that is both an input to an entity and an output from the entity.
BUFFER	Used for a signal that is an output from an entity. The value of the signal can be used inside the entity, which means that in an assignment statement, the signal can appear both on the left and right sides of the <= operator.

Figure 3

In an entity declaration, the entity name is defined (`fadder`). The name of the entity can be any legal VHDL name. The rules for specifying legal VHDL names are simple : any alphanumeric character may be used in the name, as well as the '_' underscore character. There are four caveats. A name cannot be a VHDL keyword, it must begin with a letter, it cannot end with an '_' underscore, and it cannot have two successive '_' underscores. The same rules apply to all the names in the code that are defined by the designer (e.g. port names and architecture name).

An architecture describes the internal structure/functionality of a circuit. The architecture body consists of all the statements after the word `begin`. In **Figure 2** lines `13` and `14` describe `fadder`'s functionality. Simple assignment statements are used to define the values of the outputs. Only ports characterized by one of the out, inout and buffer modes (see **Figure 3**) can be used at the left side of the "<=" operator. `S` and `Cout` outputs are assigned with logic expressions which contain the inputs `A`, `B` and `Cin`. Inputs can be found only on the right side of the "<=" operator. Logic expressions in lines `13` and `14` use parentheses to ensure the correct interpretation of the expressions. **Figure 4** shows the VHDL operators grouped into classes. Operators in a given class have the same precedence.

	Operator Class	**Operator**
Highest precedence	Miscellaneous	**, ABS, NOT
	Multiplying	*, /, MOD, REM
	Sign	+, −
	Adding	+, −, &
	Relational	=, / =, <, <=, >, >=
Lowest precedence	Logical	AND, OR, NAND, NOR, XOR, XNOR

Figure 4

The order in which the concurrent assignment statements in an architecture body appear does not affect the meaning of the code. Changing the order of the simple signal assignments (lines `13` and `14`) in **Figure 2** does not affect the meaning of the VHDL code – the same circuit is described.

A circuit is also called a design entity (or just entity) in VHDL jargon. **Figure 5** shows the general structure of a VHDL design entity.

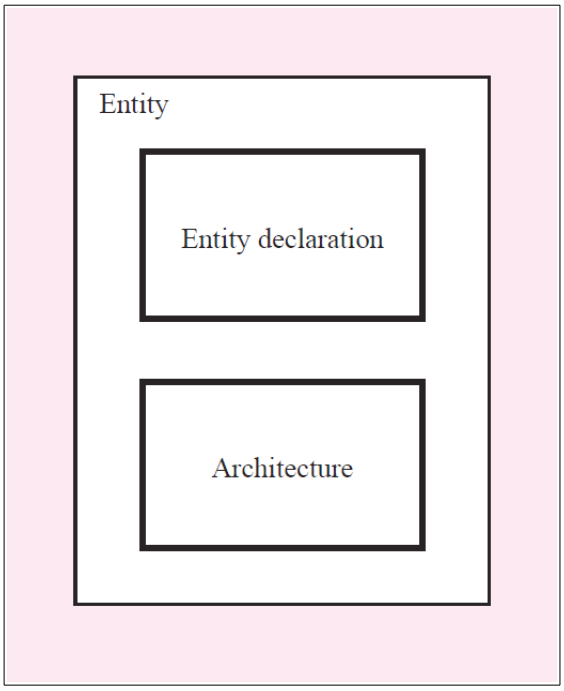

Figure 5

Selected Signal Assignment

In selected signal assignment statement a signal's value is determined by another signal's/expression's value. Using the selected signal assignment the designer can describe multiplexers, encoders and decoders. **Figure 6** shows the VHDL code for a 2:1 4-bit multiplexer.

```
 1  -- 2:1 4-bit multiplexer.
 2
 3  library IEEE;
 4  use IEEE.STD_LOGIC_1164.ALL;
 5
 6  entity mux is
 7     Port(A,B : in  STD_LOGIC_VECTOR(3 DOWNTO 0);
 8          Sel : in  STD_LOGIC;
 9          F   : out STD_LOGIC_VECTOR(0 TO 3));
10  end mux;
11
12  architecture Behavioral of mux is
13  begin
14     with Sel select
15        F <= A when '0',
16             B when OTHERS;
17  end Behavioral;
```

Figure 6

In **Figure 6** ports `A`, `B` and `F` are of type `STD_LOGIC_VECTOR`. The `STD_LOGIC_VECTOR` type is a subtype of `STD_LOGIC`, it is defined in `STD_LOGIC_1164` package and it represents a linear array of `STD_LOGIC` data objects. In line `7` the `(3 DOWNTO 0)` expression that follows

STD_LOGIC_VECTOR type indicates array's length (4 bits), the most significant bit (3) and the least significant bit (0). In line 9 the (0 TO 3) expression that follows STD_LOGIC_VECTOR type indicates array's length (4 bits), the most significant bit (0) and the least significant bit (3).

The functionality of the multiplexer is described by a selected signal assignment in the architecture body (lines 14-16 in **Figure 6**). The value of the Sel signal defines the value of the F signal. When the value of the Sel signal is '0' the value of the F signal is equal to the value of the A signal (line 15). The word OTHERS in line 16 represents all the values that the Sel signal can be equal to (because signal Sel is of type STD_LOGIC), except the logic value '0', which is already listed in line 15.

In a selected signal assignment, all possible values of the select input, Sel in this case, must be explicitly listed in the code. The word OTHERS provides an easy way to meet this requirement. OTHERS represents all possible values not already listed. In this case the other possible values are 1, Z, –, and so on. Another requirement for the selected signal assignment is that each WHEN clause must specify a criterion that is mutually exclusive of the criteria in all other WHEN clauses.

Figure 7 shows the VHDL code for a 2:4 decoder. Using the information of **Figure 7**, a 4:2 encoder (or any encoder) implementation becomes an easy task.

```
1   -- 2:4 binary decoder.
2
3   library IEEE;
4   use IEEE.STD_LOGIC_1164.ALL;
5
6   entity bdec is
7      Port(A : in  STD_LOGIC_VECTOR(1 DOWNTO 0);
8           F : out STD_LOGIC_VECTOR(3 DOWNTO 0));
9   end bdec;
10
11  architecture Behavioral of bdec is
12  begin
13     with A select
14        F <= "0001" when "00",
15             "0010" when "01",
16             "0100" when "10",
17             "1000" when OTHERS;
18  end Behavioral;
```

Figure 7

Conditional Signal Assignment

The conditional signal assignment is similar to the selected signal assignment. Using the selected signal assignment the designer can describe multiplexers, encoders/decoders and priority encoders.

Replacing the selected signal assignment (lines 14-16) in **Figure 6** with the conditional signal assignment shown in **Figure 8** (line 14) the same 2:1 4-bit multiplexer is described.

```
 1  -- 2:1 4-bit multiplexer.
 2
 3  library IEEE;
 4  use IEEE.STD_LOGIC_1164.ALL;
 5
 6  entity mux is
 7     Port(A,B : in  STD_LOGIC_VECTOR(3 DOWNTO 0);
 8          Sel : in  STD_LOGIC;
 9          F   : out STD_LOGIC_VECTOR(0 TO 3));
10  end mux;
11
12  architecture Behavioral of mux is
13  begin
14     F <= A when Sel='0' else B;
15  end Behavioral;
```

Figure 8

Likewise, replacing the selected signal assignment (lines `13-17`) in **Figure 7** with the conditional signal assignment shown in **Figure 9** (lines `13-16`) the same 2:4 decoder is described.

```
 1  -- 2:4 binary decoder.
 2
 3  library IEEE;
 4  use IEEE.STD_LOGIC_1164.ALL;
 5
 6  entity mux is
 7     Port(A : in  STD_LOGIC_VECTOR(1 DOWNTO 0);
 8          F : out STD_LOGIC_VECTOR(3 DOWNTO 0));
 9  end mux;
10
11  architecture Behavioral of mux is
12  begin
13     F <= "0001" when A="00" else
14          "0010" when A="01" else
15          "0100" when A="10" else
16          "1000";
17  end Behavioral;
```

Figure 9

The key difference between the conditional signal assignment and the selected signal assignment is that in conditional signal assignment the conditions listed after each `WHEN` clause need not be mutually exclusive, because the conditions are given a priority from the first listed to the last listed. **Figure 10** shows the VHDL code for a priority encoder.

```
1   -- Priority encoder.
2
3   library IEEE;
4   use IEEE.STD_LOGIC_1164.ALL;
5
6   entity penc is
7      Port(A,B,C : in  STD_LOGIC;
8           F     : out STD_LOGIC_VECTOR(1 DOWNTO 0));
9   end penc;
10
11  architecture Behavioral of penc is
12  begin
13     F <= "11" when A='1' else
14          "10" when B='1' else
15          "01" when C='1' else
16          "00";
17  end Behavioral;
```

Figure 10

"For Generate" Statement

The "for generate" statement is a concurrent assignment statement that is not only used in combinational circuit descriptions. It is generally used to describe circuits that exhibit regularity in their structure.

Figure 11 shows the VHDL description for an 8-bit adder using the "for generate" statement.

```
1   -- 8-bit adder with Cin, Cout and Vout.
2
3   library IEEE;
4   use IEEE.STD_LOGIC_1164.ALL;
5
6   entity adder8 is
7      Port(A,B  : in  STD_LOGIC_VECTOR(7 DOWNTO 0);
8           Cin  : in  STD_LOGIC;
9           S    : out STD_LOGIC_VECTOR(7 DOWNTO 0);
10          Cout : out STD_LOGIC;
11          Vout : out STD_LOGIC); -- 2's complement overflow indicator.
12  end adder8;
13
14  architecture Behavioral of adder8 is
15     signal tmp : STD_LOGIC_VECTOR(7 DOWNTO 0);
16  begin
17     S(0)   <= A(0) xor B(0) xor Cin;
18     tmp(0) <= (A(0) and B(0)) or ((A(0) xor B(0)) and Cin);
19
20     S1toS7 : for i IN 1 to 7 generate
21        S(i)   <= A(i) xor B(i) xor tmp(i-1);
22        tmp(i) <= (A(i) and B(i)) or ((A(i) xor B(i)) and tmp(i-1));
23     end generate;
24
25     Cout <= tmp(7);
26     Vout <= tmp(7) xor tmp(6);
27  end Behavioral;
```

Figure 11

Port `Vout` in line `11` (**Figure 11**) is the two's complement overflow indicator. In the architecture and before the word `begin` there is the declarative region. The declarative region can be used, among others, to declare signals used for internal circuit connections. Signal `tmp` is an internal signal (line `15`, **Figure 11**). Each `tmp`'s bit connects the carry out signal of each full adder to the carry in signal of the next full adder. The most significant bit of the `tmp` signal (`tmp(7)`) connects to `adder8`'s `Cout` signal (line `25`). Lines `17` and `18` calculate the least significant bit of the sum (bit `0` of signal `S`, `S(0)`) and the carry out signal of the first full adder (`tmp(0)`) respectively. Lines 20-23 hold the `for generate` statement. A `for generate` statement always begins with a label (`S1toS7 :`). Label name can be any legal VHDL name. The loop index `i` is not explicitly declared in the code; it is automatically defined as a local variable whose scope is limited to the `for generate` statement. The loop generates the logic expressions in lines 21 and 22 using index `i` in the range from 1 to 7 (`i IN 1 to 7`). Instead of the `for generate` statement, lines 21 and 22 could be repeated 7 times, using in each iteration the appropriate index number. The `for generate` results in a more compact VHDL code.

Figure 12 shows the synthesis result of the `adder8` circuit.

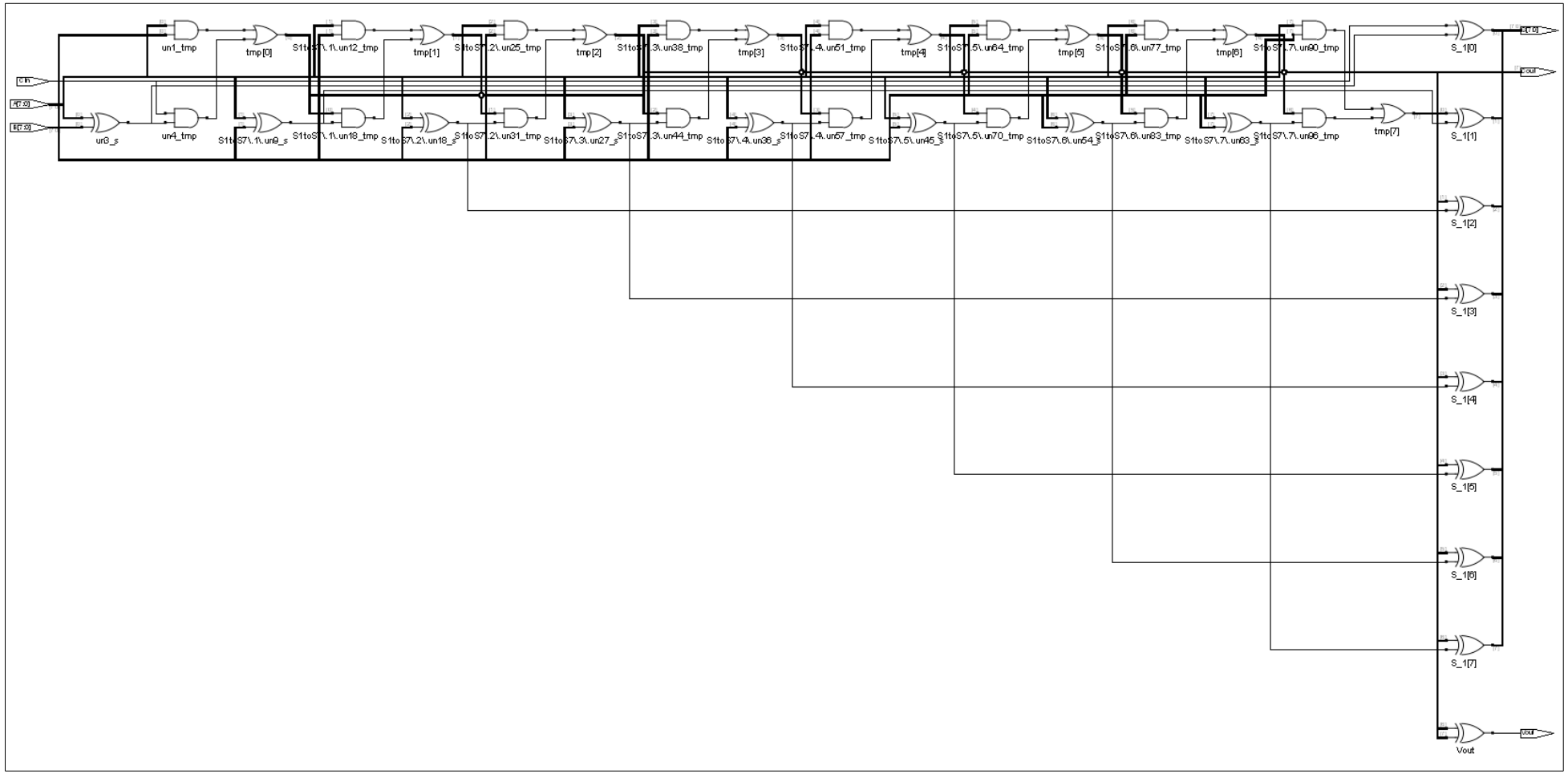
un1_tmp
tmp[0]
S1toS7\.1\.un12_tmp
tmp[1]
S1toS7\.2\.un25_tmp
tmp[2]
S1toS7\.3\.un38_tmp
tmp[3]
S1toS7\.4\.un51_tmp
tmp[4]
S1toS7\.5\.un64_tmp
tmp[5]
S1toS7\.6\.un77_tmp
tmp[6]
S1toS7\.7\.un90_tmp
S_1[0]
Cout
un9_s
un4_tmp
S1toS7\.1\.un9_s
S1toS7\.1\.un18_tmp
S1toS7\.2\.un18_s
S1toS7\.2\.un31_tmp
S1toS7\.3\.un27_s
S1toS7\.3\.un44_tmp
S1toS7\.4\.un36_s
S1toS7\.4\.un57_tmp
S1toS7\.5\.un45_s
S1toS7\.5\.un70_tmp
S1toS7\.6\.un54_s
S1toS7\.6\.un83_tmp
S1toS7\.7\.un83_s
S1toS7\.7\.un96_tmp
tmp[7]
S_1[1]
S_1[2]
S_1[3]
S_1[4]
S_1[5]
S_1[6]
S_1[7]
Vout

Figure 12

Subcircuits

A VHDL entity defined in one source code file can be used as a subcircuit in another source code file. In VHDL jargon the subcircuit is called a component. Subcircuits are used to describe modular-hierarchical designs.

Figure 13 shows the VHDL code for the `adder8` circuit (see **Figure 11**) using the `fadder` circuit (see **Figure 2**) as a component. It is assumed that `fadder.vhd` and `adder8.vhd` files are part of the same project (the files are in the same directory).

```
1  -- 8-bit adder with Cin, Cout and Vout.
2
3  library IEEE;
4  use IEEE.STD_LOGIC_1164.ALL;
5
6  entity adder8 is
7     Port(A,B  : in  STD_LOGIC_VECTOR(7 DOWNTO 0);
8          Cin  : in  STD_LOGIC;
9          S    : out STD_LOGIC_VECTOR(7 DOWNTO 0);
10         Cout : out STD_LOGIC;
11         Vout : out STD_LOGIC); -- 2's complement overflow indicator.
12 end adder8;
13
14 architecture structural of adder8 is
15    component fadder is
16       Port(A,B,Cin : in  STD_LOGIC;
17            S,Cout  : out STD_LOGIC);
18    end component;
19
20    signal tmp : STD_LOGIC_VECTOR(7 DOWNTO 0);
21 begin
22    fadder0 : fadder port map(
23       A    => A(0),
24       B    => B(0),
25       Cin  => Cin,
26       S    => S(0),
27       Cout => tmp(0));
28
29    S1toS7 : for i IN 1 to 7 generate
30       fadder1to7 : fadder port map(A(i),B(i),tmp(i-1),S(i),tmp(i));
31    end generate;
32
33    Cout <= tmp(7);
34    Vout <= tmp(7) xor tmp(6);
35 end structural;
```

Figure 13

A subcircuit must be declared using a component declaration statement. Component declaration statement is placed in the declarative region of the architecture. This statement (lines `15-18` in **Figure 13**), which is similar to the entity declaration, specifies the name of the subcircuit and gives the names and the modes of its input and output ports.

Once a component declaration is given, the component can be instantiated as a subcircuit. This is

done using a component instantiation statement (lines 22-27). A component instantiation statement consists of a label (`fadder0 :`), the component name (`fadder`) and the `port map` expression. In the `port map` expression, component ports associate with the least significant bit of `A`, `B` and `S` `adder8`'s ports, with the `Cin` `adder8`'s port and the least significant bit of the `tmp` signal (`tmp(0)`). This is called the named association. In named association the signals listed after the `port map` keywords do not have to be in the same order as the ports in the corresponding component declaration.

The `for generate` statement (lines 29-31) instantiates 7 `fadder` circuits and connects the carry out signal of a `fadder` circuit to the carry in signal of the next `fadder` circuit using the `tmp` signal. In line 30, `fadder` component instantiation uses position association; the signal names (`adder8`'s signals) following the `port map` keywords are given in the same order as in the component declaration.

Figure 14 shows the synthesis result of the `adder8` (**Figure 13**) circuit.

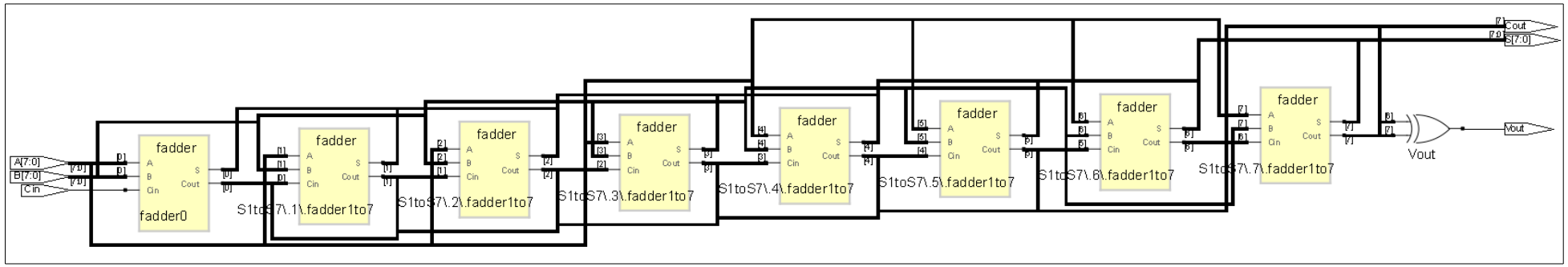
A[7:0]
B[7:0]
Cin
fadder
A
B
Cin
S
Cout
fadder0
S1toS7\.1\.fadder1to7
S1toS7\.2\.fadder1to7
S1toS7\.3\.fadder1to7
S1toS7\.4\.fadder1to7
S1toS7\.5\.fadder1to7
S1toS7\.6\.fadder1to7
S1toS7\.7\.fadder1to7
Cout
S[7:0]
Vout

Figure 14

Generic Parameters

Figure 15 shows the VHDL code for a parameterized adder.

```
1   -- N-bit adder with Cin, Cout and Vout.
2
3   library IEEE;
4   use IEEE.STD_LOGIC_1164.ALL;
5
6   entity adderN is
7      generic(N : integer := 4);
8      Port(A,B  : in  STD_LOGIC_VECTOR(N-1 DOWNTO 0);
9           Cin  : in  STD_LOGIC;
10          S    : out STD_LOGIC_VECTOR(N-1 DOWNTO 0);
11          Cout : out STD_LOGIC;
12          Vout : out STD_LOGIC); -- 2's complement overflow indicator.
13  end adderN;
14
15  architecture structural of adderN is
16     component fadder is
17        Port(A,B,Cin : in  STD_LOGIC;
18             S,Cout  : out STD_LOGIC);
19     end component;
20
21     signal tmp : STD_LOGIC_VECTOR(N DOWNTO 0);
22  begin
23     tmp(0) <= Cin;
24
25     Nadder : for i IN 0 to N-1 generate
26        parts : fadder port map(A(i),B(i),tmp(i),S(i),tmp(i+1));
27     end generate;
28
29     Cout <= tmp(N);
30     Vout <= tmp(N) xor tmp(N-1);
31  end structural;
```

Figure 15

The `generic` statement (line 7, **Figure 15**) is used to declare a parameter (`N`), which gives the designer the flexibility to make the VHDL code more general. The `N` parameter introduced in line `7` is of type `integer` and its default value is `4`. The synthesis result of the VHDL code shown in **Figure 15** is a 4-bit adder. When the adderN circuit is used as a subcircuit, the N parameter can be set at a desired value in the component instantiation statement. **Figure 16** shows the VHDL code for a 16-bit adder using the `adderN` circuit as a subcircuit. In line `24` (**Figure 16**) the `generic map` expression is used to set the value of the `N` parameter to `16`. The synthesis result of `adder16` circuit is a 16-bit adder.

```
 1  -- 16-bit adder with Cin, Cout and Vout.
 2
 3  library IEEE;
 4  use IEEE.STD_LOGIC_1164.ALL;
 5
 6  entity adder16 is
 7     Port(A,B  : in  STD_LOGIC_VECTOR(15 DOWNTO 0);
 8          Cin  : in  STD_LOGIC;
 9          S    : out STD_LOGIC_VECTOR(15 DOWNTO 0);
10          Cout : out STD_LOGIC;
11          Vout : out STD_LOGIC); -- 2's complement overflow indicator.
12  end adder16;
13
14  architecture structural of adder16 is
15     component adderN is
16        generic(N : integer := 4);
17        Port(A,B  : in  STD_LOGIC_VECTOR(N-1 DOWNTO 0);
18             Cin  : in  STD_LOGIC;
19             S    : out STD_LOGIC_VECTOR(N-1 DOWNTO 0);
20             Cout : out STD_LOGIC;
21             Vout : out STD_LOGIC);
22     end component;
23  begin
24     part : adderN generic map(N=>16) port map(A,B,Cin,S,Cout,Vout);
25  end structural;
```

Figure 16

Sequential Assignment Statements

The order in which sequential assignment statements appear in the code affects the meaning of the code. The sequential assignment statements can be used to describe either combinational or sequential circuits. The “case” and “if” sequential assignment statements are often used to describe combinational circuits.

“Case” and “If” Statements

The “case” sequential assignment statement is “equal” to the selected signal assignment; it can be used to describe multiplexers, encoders and decoders.

Figure 17 shows the VHDL code for a 2:1 8-bit multiplexer. All sequential assignment statements are can appear only in a `process` statement. The `process` statement separates the concurrent statements from the sequential statements in the architecture body. The signals shown in parentheses after the `process` keyword (in line 14) are called the sensitivity list of the `process`. For a process that describes combinational logic the sensitivity list includes all input signals used inside the process.

In VHDL jargon a process is described as follows. When the value of a signal in the sensitivity list changes, the process becomes active. Once active, the statements inside the process are “evaluated” in sequential order. Any signal assignments made in the process take effect only after

all the statements inside the process have been evaluated. The signal assignment statements inside the process are scheduled and will take effect at the end of the process.

```
 1  -- 2:1 8-bit mux.
 2
 3  library IEEE;
 4  use IEEE.STD_LOGIC_1164.ALL;
 5
 6  entity mux8 is
 7     Port(A,B : in  STD_LOGIC_VECTOR(7 downto 0);
 8          Sel : in  STD_LOGIC;
 9          F   : out STD_LOGIC_VECTOR(7 downto 0));
10  end mux8;
11
12  architecture Behavioral of mux8 is
13  begin
14     process(A,B,Sel)
15     begin
16        case Sel is
17           when '0' =>
18              F <= A;
19           when OTHERS =>
20              F <= B;
21        end case;
22     end process;
23  end Behavioral;
```

Figure 17

The “if” sequential assignment statement is “equal” to the conditional signal assignment; it can be used to describe multiplexers, encoders/decoders and priority encoders. **Figure 18** shows the VHDL code for a priority encoder.

```
1   -- 2:1 8-bit mux.
2
3   library IEEE;
4   use IEEE.STD_LOGIC_1164.ALL;
5
6   entity mux8 is
7      Port(A,B,C : in  STD_LOGIC;
8           F     : out STD_LOGIC_VECTOR(1 downto 0));
9   end mux8;
10
11  architecture Behavioral of mux8 is
12  begin
13     process(A,B,C)
14     begin
15        if A='1' then
16           F <= "11";
17        elsif B='1' then
18           F <= "10";
19        elsif C='1' then
20           F <= "01";
21        else
22           F <= "00";
23        end if;
24     end process;
25  end Behavioral;
```

Figure 18

References and Further Reading

1. Stephen Brown and Zvonko Vranesic, Fundamentals of Digital Logic with VHDL Design, 3rd Edition, McGraw-Hill, 2009

2. Volnei A. Pedroni, DIGITAL ELECTRONICS AND DESIGN WITH VHDL, Morgan Kaufmann - Elsevier, 2008

3. Edwin Naroska, comp.lang.vhdl Frequently Asked Questions And Answers, http://www.vhdl.org/comp.lang.vhdl/