Function - VHDL Example

Functions are part of a group of structures in VHDL called subprograms. Functions are small sections of code that perform an operation that is reused throughout your code. This serves to cleanup code as well as allow for reusability.

Functions always use a return statement. They are generally good for doing a small amount of logic or math and returning a result. One additional note: wait statements can *not* be used in a function.

For example, in the VHDL below there is a function f_ASCII_2_HEX below that takes as an input a 8 bit ASCII Character and converts it into a Hex value that can be used by the internal logic to do processing. This is required because letters can be uppercase or lowercase and numbers have an offset of 0x30 hex. This type of function is necessary when a UART is used on a design. The UART communicates via ASCII characters, but those need to be interpreted by the code as normal Hex Characters (0, 1, ..., E, F). Look at this ASCII table for your own reference.

The second function in this file takes an input std_logic_vector of any size and performs a bitwise exclusive or (xor) function on it. This means that it will do: r_SLV_IN(0) xor r_SLV_IN(1) xor ... r_SLV_IN(r_SLV_IN'length-1) It is versatile and can operate on input vectors of any size. This is an easy way to find the odd or even parity of an incoming signal on a UART for example.

Useful functions such as these can be wrapped up in a package file and used in multiple areas just by:

```
use library_name.package_file.all
```

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

entity example_function_advanced is
end example_function_advanced;

architecture behave of example_function_advanced is

  signal r_TEST_ASCII : std_logic_vector(7 downto 0) := X"42";
  signal r_TEST_HEX   : std_logic_vector(3 downto 0) := (others => '0');

  -- Purpose: This function converts ascii characters to hexadecimal.
  -- Numbers are 0x30-0x39 so only interpret least sig nibble.
  function f_ASCII_2_HEX (
    r_ASCII_IN : in std_logic_vector(7 downto 0))
    return std_logic_vector is
    variable v_TEMP : std_logic_vector(3 downto 0);
```

```
19      begin
20        if (r_ASCII_IN = X"41" or r_ASCII_IN = X"61") then
21          v_TEMP := X"A";
22        elsif (r_ASCII_IN = X"42" or r_ASCII_IN = X"62") then
23          v_TEMP := X"B";
24        elsif (r_ASCII_IN = X"43" or r_ASCII_IN = X"63") then
25          v_TEMP := X"C";
26        elsif (r_ASCII_IN = X"44" or r_ASCII_IN = X"64") then
27          v_TEMP := X"D";
28        elsif (r_ASCII_IN = X"45" or r_ASCII_IN = X"65") then
29          v_TEMP := X"E";
30        elsif (r_ASCII_IN = X"46" or r_ASCII_IN = X"66") then
31          v_TEMP := X"F";
32        else
33          v_TEMP := r_ASCII_IN(3 downto 0);
34        end if;
35        return std_logic_vector(v_TEMP);
36      end;
37
38      -- Purpose: This function performs a bitwise xor on the input vector
39      function f_BITWISE_XOR (
40        r_SLV_IN    : in std_logic_vector)
41        return std_logic is
42        variable v_XOR : std_logic := '0';
43      begin
44        for i in 0 to r_SLV_IN'length-1 loop
45          v_XOR := v_XOR xor r_SLV_IN(i);
46        end loop;
47        return v_XOR;
48
49      end function f_BITWISE_XOR;
50
51
52    begin
53
54      process is
55      begin
56        r_TEST_HEX   <= f_ASCII_2_HEX(r_TEST_ASCII);  -- function
57
58        if f_BITWISE_XOR(r_TEST_ASCII) = '1' then
59          report "RX Character has Odd Parity" severity note;
60        else
61          report "RX Character has Even Parity" severity note;
62        end if;
63
64        wait for 10 ns;
65
66        r_TEST_ASCII <= X"37";
67        wait for 10 ns;
68        r_TEST_HEX   <= f_ASCII_2_HEX(r_TEST_ASCII);  -- function
69
70        if f_BITWISE_XOR(r_TEST_ASCII) = '1' then
71          report "RX Character has Odd Parity" severity note;
72        else
73          report "RX Character has Even Parity" severity note;
74        end if;
75
76        wait;
77      end process;
78
79    end behave;
```