Functions, Procedures, and Testbenches

Introduction

VHDL lets you define sub-programs using procedures and functions. They are used to improve the readability and to exploit re-usability of VHDL code. Functions are equivalent to combinatorial logic and cannot be used to replace code that contains event or delay control operators (as used in a sequential logic). Procedures are more general than functions, and may contain timing controls. A testbench is a program or model written in HDL for the purposes of exercising and verifying the functional correctness of a hardware model via simulation. VHDL is primarily a means for hardware modeling (simulation), the language contains various resources for formatting, reading, storing, allocating dynamically, comparing, and writing simulation data, including input stimulus and output results.

In this lab, you will learn how to write functions, procedures, and testbenches. You will learn about the components of a testbench, and language constructs available to verify the correctness of the underlying hardware model. *Please refer to the Vivado tutorial on how to use the Vivado tool for creating projects and verifying digital circuits*.

Objectives

After completing this lab, you will be able to:
- Develop procedures for modeling a combinatorial circuit
- Develop functions for modeling a combinatorial circuit
- Develop a testbench to test and validate a design under test

Procedures Part 1

A procedure provides the ability to execute common pieces of code from several different places in a model. A procedure can contain timing controls, and it can call other procedures and functions (described in next part). A procedure is defined, within a module definition, as:

```
procedure identifier [input/output port declarations] is
      [variable declarations]
begin
      procedure statements
end identifier
```

A procedure can have zero, one, or more arguments. Values are passed to and from a procedure through arguments. The arguments can be input, output, or inout. Here is an example of a procedure definition and usage.

```
procedure HAS_PROCEDURE (
      DIN : in STD_LOGIC_VECTOR(7 downto 0);
      DOUT : out STD_LOGIC_VECTOR(7 downto 0)) is
variable k : integer := 0;
variable count : integer := 0;
begin
      for k in 0 to 7 loop
            count <= count + 1;
            DOUT(7 - count) <= DIN(count);
      end loop;
end HAS_PROCEDURE;
```

1-1. Write a procedure called calc_even_parity which will take an 8-bit number, and calculate and return number of ones. Write a module, called calc_ones_procedure, which calls the procedure with the operand received via the input port and outputs the result. Simulate the design and verify the functionality.

Functions Part 2

Functions primarily differ from procedures in that a single value is returned from executing the code. Functions are used if all of the following conditions are true:

- There are no delay, timing, or event control constructs that are present
- It returns a single value
- There is at least one input argument
- There are no output or inout arguments
- There are no non-blocking assignments

In short, functions may implement only combinatorial behavior, i.e. they compute a value on the basis of the present value of the input arguments and return a single value. Each line of code in the function is executed sequentially (they are *not non-blocking*). They are used in the right hand side of an assignment statement. Here is an example of a function definition and call. An example of a function definition in VHDL is as follows:

```
function identifier [input port declarations] return type is
      [variable declarations]
begin
      function statements
end identifier
```

To call a function, one needs to use the function identifier (with input(s) defined) as an assignment operand in a process block:

```
Func_Out <= identifier (input);
```

Here is code snippet showing how functions are used:

```
function HAS_FUNCTION (
      DIN : STD_LOGIC_VECTOR(7 downto 0);)
return STD_LOGIC_VECTOR is
      variable k : integer := 0;
      variable count : integer := 0;
      variable reverse_bits : STD_LOGIC_VECTOR(7 downto 0) := "00000000";
begin
      for k in 0 to 7 loop
            reverse_bits(7 - count) := DIN(count);
            count := count + 1;
      end loop;
return reverse_bits;
end HAS_TASK;
```

2-1. Write a function called calc_ones which will take an 8-bit number, and calculate and return number of ones. Write a module, called calc_ones_function, with one 8-bit input port and one 3-bit output port and calls the function. Simulate the design and verify the functionality.

2-1-1. Open Vivado and create a blank project called *lab4_2_1*.

2-1-3. Simulate the design and verify that the design works.

2-2. Write a procedure called add_two_values which will take two 4-bit parameters, add them, and return a 5-bit sum. Write a module, called add_two_values_function, with two 4-bit input ports and one 5-bit output port and calls the function. Simulate the design with the provided testbench, add_two_values_function_tb.vhd, and verify the functionality.

2-2-2. Create and add a VHDL module, called add_two_values_function, which defines a function called add_two_values that takes two 4-bit parameters, add them, and return a 5-bit sum. The module will have two 4-bit input ports and one 5-bit output port. It will call the function.