Using Functions

2021.1

Abstract

This lab demonstrates how to create functions in a design while coding with VHDL.

This lab should take approximately 45 minutes.

Objectives

After completing this lab, you will be able to:

- Create a new function
- Modify and use the created function

Introduction

Functions are helpful when there is a need for code repetition.

The string_utilities_sim_pkg package provides a number of string functions that are not synthesizable. This package provides additional string support through a series of constants, functions, and packages.

Functions that are supported are:

- function strAutoResize(s: String) return String – Takes the passed string and places the printing characters into a new string that is the proper length. For example, if a string were defined with a size of 32 characters and string(1) is assigned an 'A', if this were printed (using the write/writeline capabilities of TEXTIO) it would appear as 31 spaces followed by an 'A'. If strAutoResize is called first, it will return a string of length 1—just the right size for 'A'.
- function substr(str: String; startingPoint, endingPoint : integer) return String – Returns a string of the appropriate size containing the portion of the string from starting point to ending point.
- function strlen(s : String) return integer – Returns the length of the printable characters is the passed string, not the size of the string container.
- function strdel(str: String; startingPoint, endingPoint : integer) return String – Removes characters between startingPoint and endingPoint in an appropriately sized container.
- function strins(str1, str2 : String; startingPoint : integer) return String – The converse of strdel—it will insert str2 into str1 beginning at startingPoint.
- function strcat(s1, s2: String) return String – Similar to the concatenation operator '&,' but it first properly resizes both S1 and S2 before joining the two strings.

- function strpos(s : String; c : character) return integer – Returns the first occurrence of the character c within the string.

In this lab, you will be creating and adding two new functions. The first will return the position of the next occurrence of c within s, and the other will always return the last position of the last occurrence of c in s.

To practice writing synthesizable functions, you will also add two additional functions to the package: one that will 'and' all the bits of a std_logic_vector together, and another that will 'or' them together.

Understanding the Lab Environment

The labs and demos provided in this course are designed to run on a Linux platform. Many of the labs and demos can be successfully executed in the Windows environment or a native Linux environment as well.

The instructions found in this lab are expressed using the Linux notation. This includes the forward slash ('/') as the hierarchy separator instead of the Windows backslash ('\'). Students who want to run the labs directly under Windows must use the correct hierarchy separator.

Customizable environment variables enable you to tailor your environment for specific machine configurations. The only environment variable (shown below) used in the customer training environment (CustEd_VM) points to the training directory where all the lab files are located. This reduces the amount of typing you need to do when entering directory paths.

This environment variable can be customized according to your specific location and can be set for Linux systems in the `/etc/profile` file and for Windows systems by entering "env" from the search bar.

The following is the environment variable used in the customer training VM:

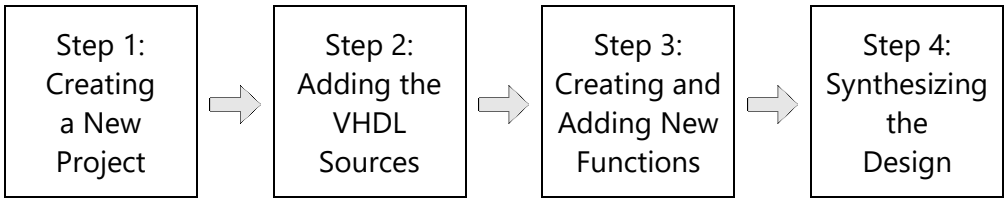
Environment Variable Name	Description
`$TRAINING_PATH`	Points to the space allocated for students to work through the labs. This directory includes prebuilt images and starting points for the labs and demos. The customer training VM sets `$TRAINING_PATH` to the `/home/xilinx/training` directory. Typically, Windows users will install the training directory under C: to keep the path names as short as possible.

Note: Environment variables are not supported from the Vitis IDE GUI. When using this tool, you must manually replace **`$TRAINING_PATH`** with the value of the variable, which in the customer training virtual machine, is **`/home/xilinx/training`**. Other tools, such as the Vivado Design Suite, will properly expand the environment variable.

Additional note about environments: Both the Vivado Design Suite and Vitis platform offer a Tcl environment. The contents of this environment are NOT preserved with the project. When the tools launch, they start with a pristine Tcl environment with none of the procs or variables remaining from a previous launch of the tools.

This means that if you sourced a Tcl script or manually set any Tcl variables and you closed the tool, when you reopen the tool (and perhaps even reopen the last project), you will need to source the Tcl script again and set any variables that the lab requires.

General Flow



Creating a New Project — Step 1

1-1. Launch the Vivado Design Suite.

If you do not recall how to perform this task, refer to the "Launching the Vivado Design Suite" section under Vivado Design Suite Operations in the *Lab Reference Guide*.

1-2. Create a new Vivado Design Suite project named *functions* and locate it in the following directory.

Browse to the `$TRAINING_PATH/functions/lab/KCU105` directory.

Target the project for the Kintex UltraScale KCU105 Evaluation Platform evaluation board.

If you do not recall how to perform this task, refer to the "Creating a Blank Vivado Design Suite Project" section in the *Lab Reference Guide*.

From the settings in the Flow Navigator, change the target language to VHDL.

Adding the VHDL Sources Step 2

In this step, you will add the VHDL sources to the project.

2-1. Add the VHDL sources.

2-1-1. Click **Add Sources** under Project Manager in the Flow Navigator.

The Add Sources Wizard opens.

2-1-2. Select **Add or create design sources**.

2-1-3. Click **Next**.

2-1-4. Click the **Plus** (+) icon and select **Add Files** from the location `$TRAINING_PATH/functions/support`.

2-1-5. Press the <**Ctrl**> and select all the files listed below:

- uart_led.vhd
- reset_bridge.vhd
- meta_harden.vhd
- uart_rx.vhd
- uart_rx_ctl.vhd
- uart_baud_gen.vhd
- LED_manager.vhd
- register8.vhd
- UART_LED_pkg.vhd

2-1-6. Click **OK**.

2-1-7. Click **Finish** to confirm adding the selected files to the project with the default associations and libraries.

2-1-8. Make certain that all the files are added to the xil_defaultlib library and not utilities_lib.

2-2. Add the simulation sources.

2-2-1. Click **Add Sources** under Project Manager in the Flow Navigator.

The Add Sources Wizard opens.

2-2-2. Select **Add or create simulation sources**.

2-2-3. Click **Next**.

2-2-4. Click the **Plus** (+) icon and select **Add Files**.

2-2-5. Select **string_utilities_sim_pkg.vhd**.

2-2-6. Click **OK**.

2-2-7. Click the **xil_defaultlib** entry in the Library field in the Add or Create Simulation Sources dialog box (highlighted in the figure below) and enter **utilities_lib**.

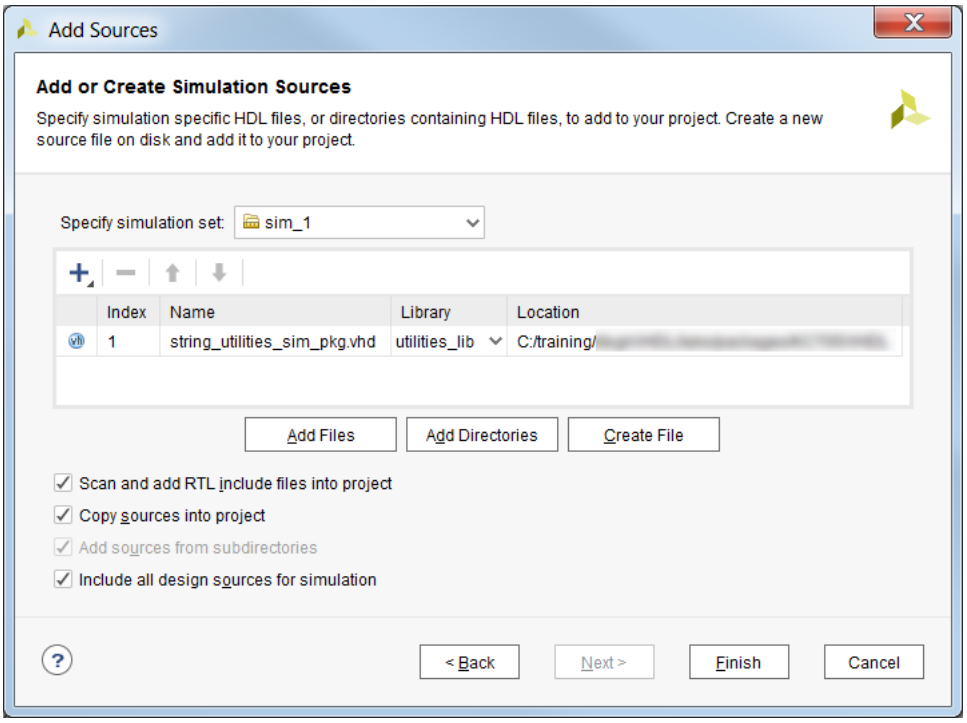


Figure 7-1: Associating the Library

The string_utilities_sim_pkg can only be used in simulation.

2-2-8. Click **Finish** to close the Add and Create Simulation sources dialog box.

Creating and Adding Functions to Existing Packages Step 3

Whenever there is a need for performing the same activity again and again, that functionality can be added in functions and these functions can be declared in the code.

Here you will be adding functions to the existing packages.

3-1. Open the existing package UART_LED_pkg for editing.

3-1-1. Select the **Sources** view.

3-1-2. Select the **Libraries** tab see all the files added to the project.

3-1-3. Expand **Design Sources** > **VHDL** > **xil_defaultlib**.

3-1-4. Double-click **UART_LED_pkg.vhd**.

UART_LED_pkg.vhd opens in the editor.

Question 1

What does this template contain?

3-2. Add the function declarations for and_bits_together and or_bits_together (takes a std_logic_vector and returns a std_logic).

Although a suggested code solution can be found in the Answers section, you should make every effort to code this yourself.

These functions take one input argument of type std_logic_vector and returns a single value of type std_logic_vector.

3-2-1. Declare the functions for *and_bits_together* and *or_bits_together*.

3-2-2. Code the outline for both and_bits_together and or_bits_together in UART_LED_pkg.vhd, using the provided templates under the package body.

Each function must return a value.

3-2-3. Define a variable of the proper type and, using a return statement, return this value.

3-2-4. Initialize this return value.

Hint: In order for and_bits_together to evaluate true, the initial value of the value being returned must be a '1'. This is not the case for or_bits_together.

3-2-5. Create an appropriate loop to perform the AND-ing and OR-ing.

Hint: Use the functions that you defined in the simulation package as a guide.

3-2-6. Select **File** > **Text Editor** > **Save File**.

Question 2

Based on the descriptions above, what must be entered into the package portion of this source file?

3-3. Open string_utilties_sim_pkg.vhd for editing and add two functions:

- **strposnext(s - string, c - character, pos - integer) where s is the string in which to locate the next occurrence of character c beginning from position pos. This function should return a non-zero integer if successful, 0 if unsuccessful.**
- **strposlast(s – string, c – character) where the position of the last occurrence of character c within s is returned. 0 is returned if c does not exist within s.**

Although suggested code solutions can be found in the Answers section, you should make every effort to code these yourself.

3-3-1. Select the **Libraries** tab in the Sources view.

3-3-2. Expand **Simulation-Only Sources**.

3-3-3. Double-click **string_utilities_sim_pkg.vhd** under **sim_1** > **VHDL** > **utilities_lib** > **Unreferenced**.

3-3-4. Write the outline for both of these functions, using either the existing functions as a template or selecting **Tools** > **Language Templates** > **VHDL** > **Common Constructs** > **User Defined Functions & Procedures** > **Function Body**.

3-3-5. Examine the code for strpos.

Notice the use of the predefined attribute ‘length to automatically adjust to the proper length of string s.

3-3-6. Code the function body for strposlast.

Hint: The last occurrence is the same as the first occurrence if the loop range is reversed.

3-3-7. Code the function body for strposnext.

Hints:

- This is the same as strpos except that the starting point is not 1.
- Both of these above functions take one string and one character as an input.
- Returns an integer value.
- A for loop can be added to cover the entire range of a string.

3-3-8. Select **File** > **Text Editor** > **Save File**.

Synthesizing the Design — Step 4

Here you will observe the newly created functions and synthesize the design.

4-1. Observe the UART_led.vhd file.

4-1-1. Observe how the UART_led_pkg package has been added in the UART_led file.

```
library xil_defaultlib;
use xil_defaultlib.UART_led_pkg.all;
```

4-2. Synthesize (compile) the design to check for any errors.

4-2-1. Click **Run Synthesis** under Synthesis in the Flow Navigator.

4-2-2. Correct errors as necessary, save the file, and resynthesize.

4-2-3. Verify that the design has been successfully synthesized.

4-2-4. Click **Cancel** if the Synthesis Completed dialog box appears.

4-3. Close the Vivado Design Suite.

4-3-1. Select **File** > **Exit**.

The Exit Vivado dialog box opens.

Figure 7-2: Exit Vivado Dialog Box

4-3-2. Click **OK**.

Some systems (particularly VMs) may be memory constrained. Removing the workspace frees a portion of the disk space, allowing other labs to be performed.

You can delete the directory containing the lab you just ran by using the graphical interface or the command line interface. You can choose either mechanism. Both processes will recursively delete all the files in the `$TRAINING_PATH/functions` directory.

4-4. [Optional] [Only for local VMs—not for CloudShare] Clean up the file system.

Using the GUI:

4-4-1. Using the graphical browser (Windows: press the <**Windows**> key + <**E**>; Linux: press <**Ctrl** + **N**>), navigate to `$TRAINING_PATH/functions`.

4-4-2. Select **`functions`**.

4-4-3. Press <**Delete**>.

-- OR --

Using the command line:

4-4-4. Open a terminal window (Windows: press the <**Windows**> key + <**R**>, then enter **cmd**; Linux: press <**Ctrl** + **Alt** + **T**>).

4-4-5. Enter the following command to delete the contents of the workspace:

[Windows users]: **`rd /s /q $TRAINING_PATH/functions`**

[Linux users]: **`rm -rf $TRAINING_PATH/functions`**

Summary

This lab walked you through the process of creating functions, both declaring and defining the functions. These functions were then added to packages.

Finally, the design was synthesized.

Answers

1. What does this template contain?

   The template contains both the package template and the package body template. Patterns are provided for types, constants, functions, and procedures declarations in the package. Functions and procedures templates are provided in the package body.

   Notice that *none* of the information is filled in — not even the package name.

2. Based on the descriptions above, what must be entered into the package portion of this source file?

   The function declaration must be entered as follows in the package declaration area:

   function strposnext(s : *String*; c : *character*; pos : *integer*) *return integer*;

   function strposlast(s: *String*; c : *character*) *return integer*;

   These can be entered anywhere within the package declaration area.

Creating a New Package

Function declaration for *and_bits_together* and *or_bits_together* functions:

```
   -- Declare functions and procedure
  function and_bits_together (signal slv : in std_logic_vector) return std_logic;
  function or_bits_together  (signal slv : in std_logic_vector) return std_logic;
```

Constant declaration for gnd_i constant:

```
     -- Declare constants
constant gnd_i : std_logic_vector(7 downto 0) := (others=>'0');
```

Function body for *and_bits_together* and *or_bits_together* functions:

```
function and_bits_together (signal slv : in std_logic_vector) return std_logic is
    variable return_sl : std_logic := 'U';
begin
    return_sl := '1';
    andAll: for i in 0 to slv'length-1 loop
                return_sl := return_sl AND slv(i);
            end loop;
    return return_sl;
end function and_bits_together;

function or_bits_together (signal slv : in std_logic_vector) return std_logic is
   variable return_sl : std_logic := 'U';
begin
    return_sl := '0';
    andAll: for i in 0 to slv'length-1 loop
                return_sl := return_sl OR slv(i);
            end loop;
    return return_sl;
end function or_bits_together;
```

Modifying the Existing Packages

Outlines for strposnext and strposlast:

```
-- ************************************************************
--  Function: strposnext
--  Inputs  : String, character, integer
--  Outputs : integer
--  Description : finds the first occurances after the pos position
--    of character in string and returns its integer position.
--    0 if character not found
-- ************************************************************
function strposnext (s   : string;
                     c   : character;
                     pos : integer) return integer is
  -- subprogram_declarative_items (constant declarations, variable declarations, etc.)
   begin
     -- function body
   end strposnext;

-- ************************************************************
--  Function: strposlast
--  Inputs  : String, character
--  Outputs : integer
--  Description : finds the last occurances
--    of character in string and returns its integer position.
--    0 if character not found
-- ************************************************************
function strposlast (s   : string;
                     c   : character) return integer is
  -- subprogram_declarative_items (constant declarations, variable declarations, etc.)
  begin
     -- function body
  end strposlast;
```

Function body for strposlast:

```
-- ************************************************************
--  Function: strposlast
--  Inputs  : String, character
--  Outputs : integer
--  Description : finds the last occurances
--    of character in string and returns its integer position.
--    0 if character not found
-- ************************************************************
function strposlast (s   : string;
                     c   : character) return integer is
  begin
      spos: for i in s'length downto 1 loop
         if (s(i) = c) then
            return i;
         end if;
      end loop;
      return 0;
  end strposlast;
```

Function body for strposnext.

```
-- ************************************************************
--  Function: strposnext
--  Inputs  : String, character, integer
--  Outputs : integer
--  Description : finds the first occurances after the pos position
--    of character in string and returns its integer position.
--    0 if character not found
-- ************************************************************
function strposnext (s   : string;
                     c   : character;
                     pos : integer) return integer is
   begin
      spos: for i in pos to s'length loop
         if (s(i) = c) then
            return i;
         end if;
      end loop;
      return 0;
   end strposnext;
```