
** (C) Copyright 2013 Xilinx, Inc. All rights reserved.

** This file contains confidential and proprietary information of Xilinx, Inc. and

** is protected under U.S. and international copyright and other intellectual property laws.

**

** / \ /

** / \ \ / Vendor: Xilinx

** \ \ V

** \ \ readme.txt Version: 1.0

** / / Date Last Modified:

** / \ \ Date Created: November 19, 2014

** \ \ / \ Associated Filename:

** \ \ \ \

**

** Device: N/A

** Purpose: Design Files for MicroBlaze IP Integrator Example Design

** Revision History: 1.0

**

**

** Disclaimer:

**

** This disclaimer is not a license and does not grant any rights to the materials
distributed herewith. Except as otherwise provided in a valid license issued to you
by Xilinx, and to the maximum extent permitted by applicable law:

** (1) THESE MATERIALS ARE MADE AVAILABLE "AS IS" AND WITH ALL FAULTS,
AND XILINX HEREBY DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS,
IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF
MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR
PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including
negligence, or under any other theory of liability) for any loss or damage of any kind or
nature related to, arising under or in connection with these materials, including for any
direct, or any indirect, special, incidental, or consequential loss or damage (including loss
of data, profits, goodwill, or any type of loss or damage suffered as a result of any action
brought by a third party) even if such damage or loss was reasonably foreseeable or
Xilinx had been advised of the possibility of the same.

** Critical Applications:

**

** Xilinx products are not designed or intended to be fail-safe, or for use in any application
requiring fail-safe performance, such as life-support or safety devices or systems,
Class III medical devices, nuclear facilities, applications related to the deployment of

XILINX INTERNAL

** airbags, or any other applications that could lead to death, personal injury, or severe
** property or environmental damage (individually and collectively, "Critical Applications").
** Customer assumes the sole risk and liability of any use of Xilinx products in Critical
** Applications, subject only to applicable laws and regulations governing limitations on
** product liability.

** THIS COPYRIGHT NOTICE AND DISCLAIMER MUST BE RETAINED AS PART OF THIS FILE AT
** ALL TIMES.

This is a simple MicroBlaze IP Integrator design that can be simulated. The stimulus for simulation comes from an elf file generated in SDK. The top level testbench only provides the clock and the reset inputs. This design has a GPIO peripheral (LEDs) that flash one after another in a sequence. The design is self-contained that is you can run simulations on this, or implement the design.

This MicroBlaze example design can be created for the AC701, KC705 and VC707 evaluation boards.

To run simulations, follow the steps outlined below:



1. In the flow Navigator, under Simulation, click **Run Simulation => Run Behavioral Simulation**.

If you want to go over the SDK flow, then follow the steps outlined below:


1. From the Flow Navigator pane in Vivado, click **Generate Bitstream** under Program and Debug.
2. If you want to test your application code on an evaluation board and/or want to generate an elf file for simulations then you need to execute the subsequent steps. This is needed so that you can download the bitstream from SDK to test your application code. ***This step is only needed if application code is to be tested on the evaluation board.***
3. Export hardware to SDK by selecting **File => Export => Export Hardware**.
4. After the hardware has been exported launch SDK by selecting **File => Launch SDK**.
5. In SDK, click on the menu item **File => New => Application Project**. The **New Project** dialog box pops up.
6. Give a name for your project. In this case we can just call it "peri_test". Leave all the other options to their default values.

7. Click on **Next**, and select **Peripheral Tests**. Click **Finish**. The application code will be automatically compiled and the elf file generated. Now you are ready to test your application code on the target hardware or modify the source code for speeding up simulations.

Running the application code on the Evaluation Board

8. Connect the evaluation board to your machine using the Digilent cable and another USB cable to the UART port and power up the board.
9. Connect to the UART by clicking on the **Terminal 1** tab and click on the **Connect**  icon to the right. Click on the **Settings**  icon to the right and make sure that **Port** is selected to the right port on your machine, **Baud Rate** to 115200, **Data Bits** to 8, **Stop Bits** to 1, **Parity** to None, **Flow Control** to None and **Timeout (sec)** to 5.
10. Select **Xilinx Tools** => **Program FPGA**. In the program FPGA dialog box make sure the Bitstream and BMM File fields are populated appropriately. Also under **Software Configuration** option, select the ELF File to Initialize in Block RAM by browsing to the peri_test.elf file for the project.
11. Click **Program**.
12. Once the FPGA is programmed and the code executes on the hardware, you should see the output "GpioOutputExample PASSED" on the terminal.

Running Simulations using the elf file generated in SDK

13. As mentioned above, the stimulus for the design comes from the software. To expedite the simulation we need to perform the following tasks.
 - a) First right click on the peri_test application and select "**C/C++ Build Settings**".
 - b) Once the "**Properties for peripheral test**" dialog box pops up, select **Tool Settings** => **Microblaze gcc compiler** => **Symbols**.
 - c) In the Defined Symbols pane click on the green + button  and add __SIM__ (i.e. underscore underscore SIM underscore underscore). Click **OK**. The reason this has to be done is that in our test example we have ifdefs for __SIM__ which expedites the simulation.
 - d) Open the file testperiph.c by selecting "src" in the Project Explorer. Find the line "status = GpioOutputExample(XPAR_AXI_GPIO_1_DEVICE_ID,8);" and right click on it. Then select "Open Declaration". This will open up the xgpio_tapp_example.c file in a separate tab.

Look for the following lines of code:

```
#ifndef __SIM__
/*
 * Wait a small amount of time so the LED is visible
 */
for (Delay = 0; Delay < LED_DELAY; Delay++);
```

```
#endif
```

This delay is enormous in simulation, so we need to add the symbol `__SIM__` to eliminate the delay as prescribed in the steps above.

- e) There are a couple of other lines of code that need to be commented out in the `testperiph.c` file, to expedite simulation. These lines of code are two `printf` statements highlighted below:

```
// print("---Entering main---\n\r");
{
    u32 status;
    // print("\r\nRunning GpioOutputExample() for axi_gpio_1...\r\n");
    status = GpioOutputExample(XPAR_AXI_GPIO_1_DEVICE_ID,8);
    if (status == 0) {
        print("GpioOutputExample PASSED.\r\n");
    }
    else {
        print("GpioOutputExample FAILED.\r\n");
    }
}
```

14. Back in Vivado, you will need to remove the existing elf file from the simulation sources and add the new elf file generated above. To remove the old elf file expand the Simulation Sources in the Sources window and select the `peri_test.elf` file. Right-click on the elf file and select **Remove file from Project**. Click **OK**.
15. You need to import the elf file and run simulations. To do this click on **File => Add Sources**. The **Add Sources** dialog box pops-up. Check **Add or Create Simulation Sources**. Click **Next** and then **Add Files**. The **Add Source Files** dialog pops up. Change the **Files of Type** to `.elf` files. Navigate `../<project_name>/<project_name>.sdk/SDK/SDK_Export/peri_test/Debug` and select the `peri_test.elf` file. Click **OK**. In the Add Sources dialog box make sure that the **"Copy sources into project"** is unchecked. This is so that, if the elf file changes for whatever reasons, the Vivado project will just pick up the new elf file as opposed to referring to the unmodified copy that it has in the project. Click **Finish**.
16. Next, you need to associate this elf file with the design. To do so, right click on `<project_name>_wrapper` in the sources pane and select **Associate Elf file**. The **Associate Elf Files** dialog box pops up. Under the Simulation sources folder, click on the browse button and navigate to the elf file that was imported from SDK. This should be at `../<project_name>/<project_name>.sdk/sdk/sdk_Export/peri_test/Debug`.
17. Now you can run simulations by clicking on Run Simulation in the Flow Navigator and selecting Behavioral Simulation. This will elaborate the design and open up the simulation window. You can run the simulation for 10,000 ns by clicking on **Run => Run For** from the menu. The **Run For** dialog box pops up. In the **Time** field, specify 10000, and in the **Units** field select Nanoseconds. Click **OK**. The simulation runs. Look for

activity on the LED_8bits_tri_o [7:0] pins. If the simulation ran properly, then you should see hex values 00, 01, 00, 02, 00, 04, 00, 08. You may have to change the radix on this signal to show hex values.