Version for EDK 10.1.03 as of January 7, 2009

Introduction

The Create and Import Peripheral Wizard is used to simplify adding user-designed peripherals into an EDK repository or an XPS project. If you wish to add a peripheral that will be connected to an OPB, PLB, or FSL bus, then this tool should make it much easier. See the Embedded System Tools Reference Manual for more information on this wizard. (Note: OPB and PLB v34 buses are deprecated. You can still use them but may have compatibility issues. If possible, please use the new PLB v46 bus instead)

This module was created before the Create and Import Peripheral Wizard was available. It remains a valuable exercise because it develops an understanding of the underlying structure of an XPS project and provides insight into how to handle a peripheral that might not exactly fit the model of a typical peripheral. In this case, the core to be added also has a connection directly to the processor, not just to the bus. Finally, it provides an example of how to design a peripheral that connects directly to the OPB bus without Xilinxs IPIF layer and IPIC interface as an intermediary.

Goals

- Add a user-designed peripheral to a basic MicroBlaze system. You will be provided with a core called snoopy that is written in VHDL.
- Demonstrate the required structure necessary for interfacing user-designed cores to the Xilinx cores in an XPS project.
- Learn about how you would make your own core to attach to the PLB.

Requirements

Module m01: Building a MicroBlaze System in XPS

It is suggested that modules m02, m03, and m04 also be completed before proceeding.

Preparation

- Review the handout outlining the EDK project structure provided in Module m01. We will be focusing on the `pcores` subdirectory for this module.
- If you are unfamiliar with the profiling of code, read the manual page for "gprof", which is accessible on most UNIX systems via the "man" command and directly from the FSF.

Background

To this point, you have only been adding cores from the existing library. If you cannot find a core with the required functionality, you will have to add your own.

This lab adds a simple core that can be used to profile the code running on a MicroBlaze. This has similar functionality to the "gprof" utility for profiling that is included in the GNU toolchain. However, it is much more accurate. You may find it useful for your project.

Step-by-step

1. Copy your working `lab1` project into a new directory called `lab5`.

2. Delete `lab5/code/lab1.c` — you don't need it any more.

3. Unzip `m05.zip` into the directory containing `lab5/`. This will add a new program (`lab5/code/system.c`), the custom peripheral (`lab5/pcores/snoopy_v1_00_a`), and some supplemental files.

4. Take time to look through the directory structure of the snoopy core. The naming structure is essential for XPS to be able to detect a user's peripheral. All user cores must be located in the `pcores` subdirectory or in a globally specified path to a peripheral repository. User cores can be defined using Verilog, VHDL, or a mixture of the two.

   Along with the HDL files used to implement the core, the user must also include data files: a `.pao` file (Peripheral Analyze Order) and an `.mpd` file (Microprocessor Peripheral Description). The former lists the order in which files in your design should be synthesized to resolve component architectures. The latter describes the external interface of the core to a system. For more information on these files and their structure, refer to the Embedded System Tools Guide.

   Notice that the version numbers in the core name and the version numbers in the data file names differ (*e.g.*, version 1.00.a of the snoopy core includes a data file called `snoopy_v2_1_0.mpd`.) The version in the core name reflects the core's version. The version in the data file names reflects the version of the syntax used to write the data file.

   The easiest method for including user IP in an EDK project is to follow an example. When you develop your own cores for your project, you can use snoopy as a guide. The cores provided by Xilinx in the `$XILINX_EDK/hw/XilinxProcessorIPLib/pcores/` directory may also be used as a reference.

   You are also encouraged to look around through the directories in the EDK installation because there is a lot of source code available that might help you or guide you with your own designs. Just as you learn to write good prose by reading good prose, you learn to code well by reading good code. HDL for IP core can be found in the `$XILINX_EDK/hw/XilinxProcessorIPLib/pcores/` directory and C code for the drivers can be found in `$XILINX_EDK/sw/XilinxProcessorIPLib/drivers/`. Documentation for the IP cores and drivers can also be found in these directories.

5. The snoopy core is a snooping profiler that is able to profile software running on a soft processor in real time. The counters calculate the exact number of clock cycles spent executing contiguous address ranges. The user specifies the number of counters and the lower and upper bounds for each counter before synthesis. This information can be used by embedded system designers to determine which, if any, sections of the software should be moved to hardware to achieve the required design specifications.

6. Lauch XPS and open your `lab5/system.xmp` project. Open the IP Catalog tab located in Project Information Area. The snoopy core should appear filed under Project Repository. Add an instance of the snoopy core by right clicking the core and selecting Add IP. The core will appear in the list of System Assembly View. Connect it to the slave PLB by clicking the empty gold circle associated with the core. Switch to the Addresses view and place the new snoopy instance somewhere in the memory map. The core requires 256 bytes of address space to be allocated to it aligned at a 256-byte boundary (don't forget that the PLB only covers the address space *not* covered by the ILMB and DLMB — *i.e.*, 0x00010000-0xFFFFFFFF). Feel free to check the parameter settings for the core by double-clicking on it. The default values are fine for now.

7. The snoopy core functions by monitoring some trace signals provided by the MicroBlaze. These are documented on the MicroBlaze core's datasheet, accessible by right-clicking on the microblaze_0 entry in the System Assembly View and selecting View PDF Datasheet. Specifically, the `PC_EX` signal of the snoopy core must be connected to the `Trace_PC` signal provided by the MicroBlaze and the `valid_instr` signal of the snoopy core must be connected to the `Trace_Valid_Instr` signal of the MicroBlaze. Wire up these connections by entering `Trace_PC` and `Trace_Valid_Instr` in the Net column next to the snoopy core's `PC_EX` and `valid_instr` ports, respectively.

8. Select the Application tab, remove the old system source file, and add the new source file (`lab5/code/system.c`). Double-check that the compiler options are set to build the user application without optimizations (*Hint: look at the Optimization Level in the Debug and Optimization tab of the Compiler Options dialog*) and the libraries with optimizations (*Hint: look for the extra_compiler_options variable in the Software Platform Settings dialog*). From the Software menu, select Build All User Applications to regenerate the libraries and compile the software applications. **Note:** this will not regenerate the hardware components of your design.

9. Launch an EDK shell and disassemble your executable into a file. Open the disassembled file to determine the address ranges you will profile. You will be selecting contiguous address ranges to profile based on function calls. Double click the snoopy_0 entry in the System Assembly View to edit the parameters. Profile the following functions:

Counter	Function
0	_start
1	exit
2	_crtinit
3	main
4	_exception_handler
5	_interrupt_handler
6	_program_clean
7	_program_init
8	print
9	putnum
10	outbyte
11	XUartLite_SendByte
12	XUartLite_RecvByte
13	All Functions (complete program)

   The lower bound should be the starting address of the function and the upper bound should be the address of the last instruction in the function. Change the value for the `NUM_COUNTERS` parameter to 14.

10. Generate the bitstream, download it to the FPGA, and launch XMD.

11. To reset the counters, use the memory write command in the XMD window: `mwr reset_address <value>`, where the `reset_address` is any address in the snoopy core's allocated address space. The counters are designed to reset regardless of the written value.

    The `show_count.tcl` script you unzipped into your project directory can be used to read the counters. Type `set snoopy_c_baseaddr <value>` to set the base address of the snoopy peripheral. Set the `snoopy_num_counters` variable to the correct value the same way. Finally, type `source show_count.tcl` to display the contents of the 64-bit counters. As you have just reset the counters, all the values should be zero.

12. Open the GDB Debugger and connect to the target. The assembly listing of the main routine should be visible, with the program counter set to 0x800. Set a breakpoint at the closing bracket of main and execute the program.

    After you have run your application, you can check the new counter values again using the `show_count.tcl` script. Are they all non-zero values? Why or why not? (*Hint: look at the disassembled code to understand the values.*)

13. Reset the counter values as previously instructed and rerun the program. Run the `show_count.tcl` script again and look at the values of the counters. Are they all the same as last time? Why or why not? You should compare your results to those found in the file `example_results.txt` to verify the counter values. Does your profiling data explain where all the execution cycles were spend (*i.e.*, does the total number of cycles executed equal the sum of the cycles spent in the functions profiled)?

14. You can also use the counters to determine how many clock cycles are required to execute a single instruction by setting both the upper and lower bounds to the same address. If the instruction is executed only once, the number of clock cycles should be the same as what is specified in the MicroBlaze processor manual. Familiarize yourself with the available XMD commands or change the core parameters to better understand how your application runs on the MicroBlaze processor.

15. If you decide to make your own core by hand then the snoopy core can be used as an example. Alternativley, you can use the Create and Import Peripheral Wizard (located under the hardware menu of XPS) to create a template for a new IP core. The wizard will generate a working example core that connects to your choice of interface (PLB, FSL, OPB, etc) that you can experiment with.

Look at Next

Module m06: Using ISE
Module m08: Using External ZBT Memory
Module m10: Fast Simplex Link (FSL) Interfaces
Module m12: Introduction to ChipScope
Module m13: Real-time MP3 Decoder Implementation Using the MicroBlaze Soft Processor