Adapting the VirtexII Version of the Picoblaze soft processor for use in a SpartanII target device.

Author : Nial Stewart
V1.0 30/01/03

Xilinx provides different 'flavours' of the Picoblaze soft core processor with architectures optimised for different target device families.

The version targeted at the SpartanII family has an address space of just 8 bits limiting the number of instructions to 256 (ie one SpartanII BlockRam). The version provided for the VirtexII family uses an address space of 10 bits (one VirtexII BlockRam 1024x18) to provide 1024 instructions.

Although not architecturally efficient, the VirtexII Picoblaze version can be adapted to be used in a SpartanII device using 5 SpartanII BlockRams. This conversion might be benefitial in cases where the 256 address instruction space is a limiting factor.

This is a discussion paper on how the VirtexII Picoblaze may be used in a SpartanII device, using an very simple application running on a BurchED B5-X300 board as an example. Some knowledge of the Picoblaze processor and web-pack is assumed, see the Quickstart tutorial for more details. The example project requires Perl and the Xilinx Web-pack to be installed.

Conversion Discussion

There are two technical obstacles with using the Picoblaze2 with a SpartanII architecture.

The first is the size of the BlockRam used to store the instruction code, this uses one VirtexII Blockram in 1024x18 configuration. This can be replaced by 5 SpartanII Blockrams in 1024x4 configuration, but the ram initialisation attributes which define the code to be run must be translated from one large ram into 5 sets of initialisations for the 5 smaller rams. Initially this appeared a fairly complex problem, the VirtexII initialisation is split into 2 sections, one for the main block and one for the 2 'parity' bits (which give the ram the unusual 18 bit width). Luckily the Assembler produces a 'design_name.hex' file which is a simple list of 5 hex characters for each address. This is easily split into configuration data for the 5 SpartanII Blockrams used.

A Perl routine (***vir2spart.pl***) was written to read in 'design_name.hex' and create 5 sets of initialisation parameters. A template for the output file ***template.vhd*** is read in and the initialisation parameters are inserted. The resulting file is always output as ***picocode.vhd*** so this is the only component instantiation needed with the Picoblaze2 if different sets of code are being tested. A Dos batch file '***comp.bat***' was written to run the assembler ***KCPSM2.exe***, run vir2spart.pl on the output hex file and copy the resultant ***picocode.vhd*** to the VHDL subdirectory. This is run by entering 'comp design_name' **without** the .psm appended.

NOTE: There is no error checking in comp.bat or vir2spart.pl so a check for errors should be made each time a design is run.

The second technical problem with the conversion to a SpartanII architecture is the use of a distributed dual port 32x1 ram structure (RAM32x1D). This is not supported in the SpartanII so it must be replaced with two 16x1 distributed rams. Luckily the RAM32x1D is only used in one module data_register_bank. This is commented out and the supplied ***data_register_bank.vhd*** is added to the project source to replace it.

Example Project
Setting Up

Picoblaze2 Download

The Picoblaze2 should be downloaded from the Xilinx web site..

http://www.xilinx.com/ipcenter/processor_central/picoblaze/index.htm

Download the app note and reference design for 'Picoblaze for VirtexII and VirtexII Pro FPGAs' (xapp627.pdf and xapp627.zip). Unzip xapp627.zip into a temporary directory.

Example Project Download

The example project should be downloaded from the downloads page of Nial Stewart Developments..

http://www.nialstewartdevelopments.co.uk/

This should be unzipped to a 'project' directory location. This is contains three subdirectories

VHDL – Source files
SW – Assembly files, the assembler KCPSM2.exe and an Perl routine to generate SpartanII instantiations.
XILINX – Webpack will be run in this directory.

The Picoblaze2 ***kcpsm2.vhd*** should be copied from the temporary Xilinx download directory to the VHDL directory. **This must be edited** to comment out the vhdl module 'data_register_bank'. As discussed above, this contains an instantiation for a distributed ram structure that's not supported on the SpartanII.

KCPSM2.exe, ***ROM_form.vhd*** and ***ROM_form.coe*** should be copied from the temporary Xilinx download directory to the SW subdirectory.

Building the Example

The example project is a very simple routine used to verify the software is being buit correctly. A main loop constantly outputs a 'count' value to the LEDs, with an interrupt routine incrementing 'count'. ***Pico2test.vhd*** is a wrapper round the Picoblaze, code ram (instantiated as picocode) and an output port mux steering the output on port4 to the LEDs.

A .ucf file is provided with the example project. This assumes a BurchED B5-X300 board is being used with a B5-LEDS module attached to header D. The clock constraint is set to 50MHz, but the P+R effort is left to the default low.

Open a Dos command prompt and navigate to the /SW directory. Type 'comp testcode', hit return and check that the assembler finishes sucessfully (KCPSM2 sucessful, KCPSM2 complete). The assembley code ***testcode.psm*** has now been assembled, converted to SpartanII blockram format and copied to the /VHDL directory.

In Web-pack create a new project in the /Xilinx directory. Add ***pico2_test.vhd***, ***kcpsm2.vhd***, ***picocode.vhd*** and ***data_register_bank.vhd*** as sources. Verify that in properties of the 'Generate Programming File' process, the 'Start-Up Clock' under 'Startup Options' tab is set to 'JTAG Clock'.

Build the project (the .ucf should be picked up automatically) and programme the board. When the board's sucessfully programmed LED0 should be lit, and upon each key press the LEDs will provide and incrementing binary count.

To demonstrate verify that ***data_register_bank.vhd*** has been incorporated and is working correctly a second piece of assembly has been provided ***reg_test.psm***. This resets all registers to 0 then adds varying amounts to the registers, alternating between the upper and lower banks and keeping a running total. The result should be 0x78.

To try this code, type 'comp reg_test' in the Command window in the /SW directory. When this has run right click on the 'Generate Programming File' process in Webpack and select 'Rerun All'. When the design has built (a matter of seconds) re-programme the board and check that the LEDs represent the hex value 0x78.

These are trivial examples that don't demonstrate the capabilities of the Picoblaze but by experimenting with the Address assembler directive it can be shown that all 1024 addresses can be used sucessfully.