

Quick Start Guide to using the Xilinx Picoblaze 8 bit Microcontroller on the BurchED B5-X300 Board.

Author : Nial Stewart

15/01/03

Increasingly FPGAs are being used with embedded CPU cores. Xilinx and Altera provide the 32 bit Microblaze and 16/32 bit NIOS cores respectively, but to use these development systems must be purchased for approximately \$1000.

Xilinx also provides the minimal architecture 8 bit Picoblaze CPU free. As Xilinx describes it.. "PicoBlaze and its derivatives are fully customizable 8-bit soft microcontroller macros that provide 49 different 16- to 18-bit instructions, 8 to 32 general-purpose 8-bit registers, 256 directly and indirectly addressable ports, reset, and a maskable interrupt."

This is a quick start guide to briefly describe the Picoblaze and to step through getting a simple implementation running on a BurchED FPGA evaluation board (B5-X300, although older BurchED boards could also be used). The B5-Led module is used to output the results (the B5-7Seg-Display could possibly be substituted). The Xilinx Web pack is used for Synthesis, P+R and to programme the board. No previous experience with Web-pack is assumed, so step by step build details are included.

CPU Download

Please download the Picoblaze core reference design and documentation from...

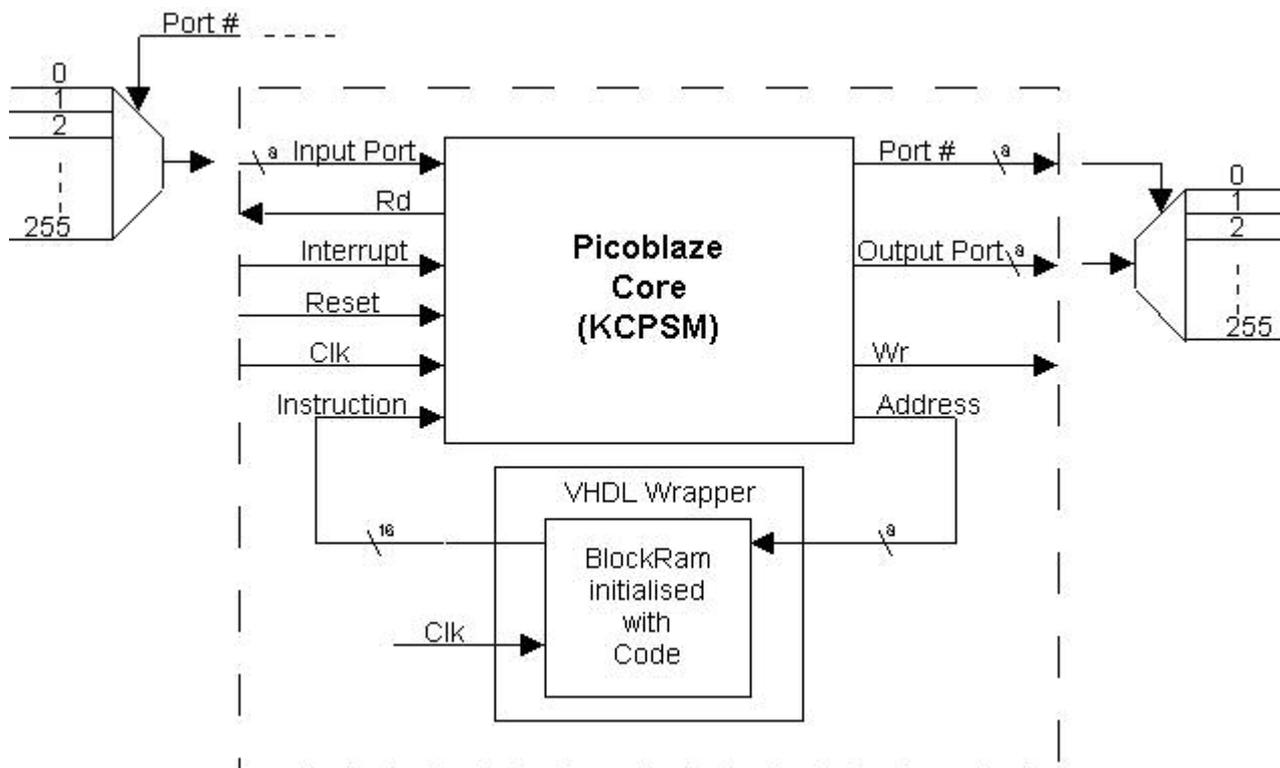
http://www.xilinx.com/ipcenter/processor_central/picoblaze/index.htm

You will have to create an account and log in, this is free and fast. Download the versions for SpartanIIE (xapp213.pdf and xapp213.zip).

Unzip the reference design to a temporary location, there's more than is needed to get started and everything's bundled in one directory.

CPU Overview

The structure of the Picoblaze is shown here.



The instructions for the processor are held in a 256x16 Blockram which is initialised to contain the firmware. When the assembler is run it outputs a VHDL wrapper round the a blockram which contains the Xilinx INIT directives, and hides all ports except an Address and Clk in and Instruction out.

All input/output to the pico blaze is via an input and an output port. To differentiate different ports an input multiplexer and output de-multiplexer must be used. The Rd and Wr strobes are be used to register data in and out (use of the Rd strobe is optional).

The Interrupt input is level sensitive but must be enabled by the firmware before it will be acted upon. The firmware can enable or disable it at any time.

Example Application

As an example application with the Picoblaze we'll build a simple binary counter. This will increment every time the Test Switch is pressed using an interrupt routine to increment a count, the main body of code will output the count value to LEDs when it has changed.

To keep this project better organised it has been split into 3 separate directories..

VHDL – This contains the Picoblaze (*kcpsm.vhd*) and the top level wrapper which connects it to the ram etc.

SW – This is where the assembly (*picocode.psm*) is assembled into the vhd wrapper (*picocode.vhd*).

XILINX – This is where we'll run the Web-Pack for synthesis, P+R and to program the FPGA.

Copy *kcpsm.vhd* (the picoblaze core) from where you unzipped the Xilinx download to the vhd directory, and *kcpsm.exe* (the assembler), *ROM_form.coe* and *ROM_form.vhd* into the SW directory.

The Program

If you look in the SW directory which you've unzipped you'll find the Assembly file *picocode.psm* which is used in this example. Please refer to *xapp213.pfd* for full details of the picoblaze instruction set.

The code starts by declaring a constant for two different port numbers, and renames two registers to **count** and **old_count**. The interrupts are enabled and initial values for the two registers are set.

The main loop checks to see if **count** has incremented by continually subtracting **old_count** and jumping back if the result is zero. If the result isn't zero the new value of count is output to the **leds_port**, the new value of **count** is stored in **old_count** and a jump is made back to the start of the loop.

The start address of the interrupt handler routine is set by **Address B0** (this is optional, if it wasn't set the interrupt routine would follow straight after the main bulk of code). The first thing the interrupt handler does is increment **count**. There follows a software delay to stop multiple interrupts due to contact bounce in the switch before returning from the interrupt handler. This return also re-enables the interrupt.

Please refer to *xapp213.pdf* for further details of the Picoblaze architecture and instruction set.

Assembling the code

Open a DOS window and navigate to the /SW directory of your project. At the prompt type "kcpsm picocode" and hit return. The assembler quickly assembles the code and generates the VHDL wrapper using *ROM_form.coe* and *ROM_form.vhd*, hopefully finishing with "KCPSM Successful" and "KCPSM Complete".

The output file is *Picocode.vhd*, you will see this is simply a wrapper round a RAMB4_S16, with the contents initialised to contain the assembled code.

Picocode.vhd is then instantiated directly in the application.

Top Level Wrapper

Looking in the /VHDL subdirectory you will see the top level file, *Pico_test.vhd*.

This connects the instantiation of the Picoblaze (kcpsm.vhd) to application code (picocode.vhd). It also generates a basic reset signal, generates an interrupt on the falling edge on the intrpt_n input and steers outputs on any port except port2 to the LEDs.

Setting up the Board

The .UCF file that is supplied in the /Xilinx directory is configured for the LED module to be connected to header D. The clock configuration jumpers should be set for a 20MHz clock.

Building the Project

Run up the Xilinx Web-Pack and select File->New Project. Enter whatever you want to call the project and change the location to the /Xilinx directory. Ensure that the Device Family and Device are selected properly and that Design Flow is set to XST VHDL (click in the boxes to change the selected values).

Select Project->Add Source then navigate to the /VHDL directory and select *kcpsm.vhd* and *pico_test.vhd* then confirm they are VHDL modules. Web-Pack analyses the designs and displays the structure in the left hand pane. Now select Project->Add Source, navigate to the /SW directory and select *picocode.vhd* again confirming that it's a VHDL module.

In the middle pane 'Processes for Current Source' right click on "Generate Programming File" and select Properties. Under the "Startup Options" tab change the Start-Up Clk to JTAG Clk then hit OK.

The .UCF file for the project (*pico_test.ucf* in the Xilinx directory) contains pin assignments for the B5-X300 board. If a different board is being used this should be edited to reflect the different pin assignments. As this is a simple demonstration with minimal functionality, and not particular constraint on driving the LED outputs, we will set a clock constraint of 50MHz (20ns).

Make sure that Pico_test is highlighted in the left hand pane then right click on "Generate Programming File" and this time select Rerun All. This forces Web-pack to run through the complete device build from Synthesis to P+R and finally the generation of the programming file. Open up the process hierarchy by clicking on the + symbols in the middle pane to follow progress more closely.

Programming the Device

When the build has completed (this will only take a minute or so) ensure that the BurchED board is powered up and programming cable is connected to the PC's parallel port. Double click on "Configure Device (IMPACT)", and when the programmer window opens it should detect the target device. Right click on the device graphic and select 'Program..', then hit OK in the 'Program Options' window.

If the device programs successfully this will be reported and LED(0) should be lit (the code initialises the count to 1).

Testing

Pressing the 'Test Switch' on the board should increment the count, reflected by the state of the LEDs.

When testing the board, it might appear that more than one count is occurring per key press. This is due to the software delay in the interrupt routine not being long enough to eliminate contact bounce problems. Experiment with the delays in the interrupt routine to see how they affect this problem. Remember to re-assemble the code before re-building the project. That's it ☺.

Discussion and Further Development

As this is a simple design we have not looked at simulation. The Picoblaze core is provided as a VHDL module, and as the assembler output is VHDL, the system can easily be simulated for complete verification.

Although the Picoblaze is a small implementation of a microprocessor, limited to 256 instructions, it's small size means that larger tasks can be split into smaller tasks which can be assigned to a number of different processors. The SpartanIIE-300 that's used on the B5-X300 board has 16 Blockrams, so in theory could hold 16 different instances of the Picoblaze. It would also be easy to design modules round a Picoblaze implementation to perform the 'donkey work' with the Picoblaze tying everything together. An example is the UART modules provided with the Xilinx reference download.

The instruction memory may also be increased in size by using an output port bit to 'bank switch'. A utility, ***PSMSPLIT.exe***, is included in xapp213.zip to generate a VHDL code wrapper including two blockrams and a 'switch' input. This is discussed further in xapp213.

The fact that the Picoblaze code is stored in BlockRam which has dual port capabilities gives us some flexibility. Modifying the VHDL code wrapper would allow access to re-programme the Picoblaze in-situ. This could be easily done with a state machine, or perhaps another instance of a Picoblaze. The new ram contents would have to be extracted from the assembler output .vhd file, this could be done with a simple Perl routine etc... The processor being re-programmed should be held in reset until the new contents have been programmed.

An expanded version of Picoblaze is available for the VirtexII series of FPGAs. As the blockrams of these devices are bigger, the standard code size is 1024 addresses of 18 bit instructions. There are also 32 registers. This version could probably be adapted for use with the SpartanII devices by splitting the code ram into 5*SpartanII BlockRams (5 of 1024x4 bits). This is a slightly less than optimal implementation, but will provide a much more capable processor for little effort (although some work would need to be done on the new initialisation values).

The constraint for this example was set to 50MHz as a 'suck it and see' exercise. Although the FPGA has almost no content (2% os slices used), the results of a P+R can be as good as 17.2ns worst case timing on the clock constraint of 20ns. This was routed with the P+R effort set to the default mimimum effort, so these results would tend to back up the claim in xapp213 that the Picoblaze can run at up to 40MIPS.