Picoblaze Sample Design
A Quick-Start Guide Using the Digilent NEXYS 2 Spartan 3E Kit

Doug Wenstrand

www.echelonembedded.com

A Picoblaze Based Sample Design for Digilent NEXYS 2 Board

Introduction:

In the process of an FPGA design, the engineer is often confronted with tasks which seem more well suited to software than hardware state machines. One great solution to this problem is a tiny embedded soft-core processor resident inside the FPGA. The Xilinx Picoblaze processor is an excellent fit for small designs, as it occupies only 96 slices , which is 1% of even a relatively small Xilinx XC3S500E. The use of a small processor like the picoblaze enables several things in an engineers design. First, it allows the engineer to write software for tasks which seem more straightforward to code that way. Second, it can replace other small processors that might be in the FPGA-based system with minimal impact on the FPGA design itself. Finally, and perhaps most importantly, it allows for changing behavior in a flash, rather than waiting for a time consuming Synthesis / Place / Route FPGA flow. This tutorial will lead the student through a sample picoblaze design which shows how to design both the hardware and software, and load it all to the FPGA. The student will then modify the behavior of the machine (via changing software), and update that code within the FPGA in seconds for rapid testing.

Project Description and Location

The sample project is written for the Digilent NEXYS2 Spartan 3E Starter Kit. This design is easily modifiable to other Xilinx platforms, as all the source is included; however the pre-generated pin assignments, and detailed programming instructions will not be relevant. The project provides example interfaces to some of the simple peripherals on the board:

1) 8 LEDs
2) A 4-Digit Seven Segment Display
3) 8 Slider Switches
4) 4 Push-buttons
5) RS232 Port (115200, 8-N-1, no flow control)

The project is located here : NEXYS_PBLAZE

Downloading and Testing

To get started, first unzip the archive to a location without any spaces in the full path (i.e. not the desktop). Once the project is unzipped, test the project and your hardware to see the end result. To do this, either load the bitfile from Xilinx IMPACT using your own cable, or from the Digilent ExPORT tool over the USB Cable. The bitfile : toplevel.bit is located along with the ISE project at the root project directory. Obviously, those without the NEXYS2 should skip this step, and proceed to rebuilding the project. Once the device is programmed, the picoblaze inside the FPGA should start running the test program, which:

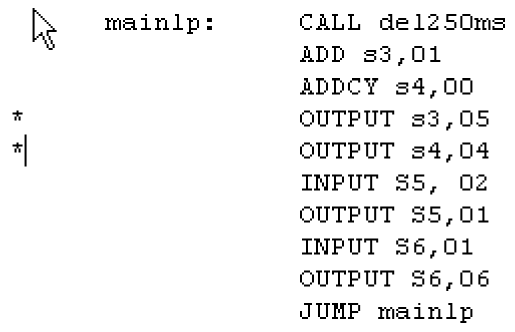
1) increments the 4 characters on the 7 segment display as 1 hexadecimal number at a rate of 3-4Hz
2) At the same rate, reads the slider switches, and sends their value (in raw binary) to the RS232 port.
3) Similarly, the picoblaze checks the incoming UART for characters, and writes their value to the 8 individual LEDs on the board.

A handy terminal program for observing this program, *Bray's Termina,l* is located Here. This program will allow you to communicate in raw hexadecimal rather than ASCII

Changing Code

Once you know the hardware and sample software works on your platform, it is a good time to demonstrate how quickly code changes can be made (especially when compared to an alternative like a VHDL state machine).

To do this, navigate to the KCPSM3\Assembler subdirectory. The program, named testprg.psm can be opened and edited with a standard text editor. For this example, we will byte swap the 4 digits on the 7 segment display.



```
     mainlp:     CALL del250ms
                 ADD s3,01
                 ADDCY s4,00
*                OUTPUT s3,05
*                OUTPUT s4,04
                 INPUT S5, 02
                 OUTPUT S5,01
                 INPUT S6,01
                 OUTPUT S6,06
                 JUMP mainlp
```

Note the reversal on the two lines marked with an *

Now, the KCPSM3 assembler should be run, which will assemble the design, and modify the VHDL file *testprg.vhd*. This VHDL file is part of the top level ISE project, and will be included on the next Synthesis/Place/Route and thus will run with normal FPGA configuration. Additionally, it makes a *.hex* file, which will allow us to test much sooner. With a simple call to hex2svf.exe, the program can be converted into an SVF file, which can be sent via JTAG to the picoblaze program memory inside the xilinx. The picoblaze is then reset, and will run the code immediately, without any change to the hardware surrounding it. A batchfile "Make.bat" is included in that directory, which will create a single-click solution for making the desired file for JTAG loading *testprg.svf*. Load this file to the FPGA via Digilent ExPORT (part of the ADEPT suite), and the picoblaze will be running new code in seconds. Verify that in fact you have byte swapped the outputs to the display. **Note, if you are not using the NEXYS2 board, instructions for loading**

the svf file via Xilinx cable (from the command prompt) are available in the Documentation in the JTAG_Loader subdirectory.

Changing Hardware

Clearly, one of the benefits of the picoblaze is the ability to hook it with custom hardware (the main reason for the FPGA in the first place!). In this section, the hardware interface to the picoblaze will be discussed by looking at the *toplevel.vhd* file in ISE. It can easily be seen that adding extra devices to which the picoblaze can talk is a trivial matter. Note: for details on the picoblaze instruction set, and hardware interface, the KCPSM3 folder contains some excellent documentation courtesy of Xilinx.

Writeable Devices

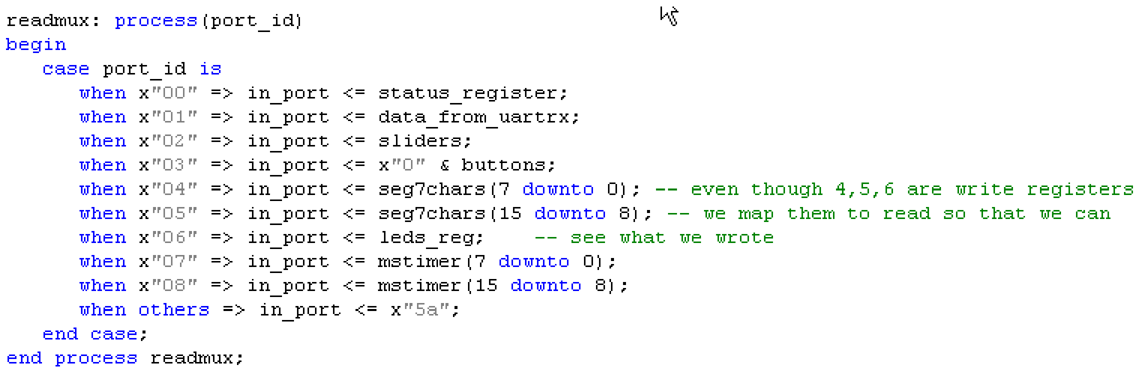
Mapping devices which are written to from the picoblaze is a very simple matter. The picoblaze provides an address (PORT_ID) which allows up to 256 output devices to be simply mapped. Deciding when to latch the data is as simple is looking at the clock, the port_id, and a 1 clock wide write_strobe from the processor. Consider the 1 line of code it takes to create and output register for storing LED values:

```
leds_reg <= out_port when rising_edge(clk) and (port_id = x"06") and (write_strobe='1');
```

This creates 8 d-flipflops, and enables them for writing only when the picoblaze is writing to port_id 6.

Reading Devices

To set the picoblaze to be able to read from multiple devices, a simple multiplexer is perhaps the most straightforward way.



```
readmux: process(port_id)
begin
   case port_id is
      when x"00" => in_port <= status_register;
      when x"01" => in_port <= data_from_uartrx;
      when x"02" => in_port <= sliders;
      when x"03" => in_port <= x"0" & buttons;
      when x"04" => in_port <= seg7chars(7 downto 0); -- even though 4,5,6 are write registers
      when x"05" => in_port <= seg7chars(15 downto 8); -- we map them to read so that we can
      when x"06" => in_port <= leds_reg;     -- see what we wrote
      when x"07" => in_port <= mstimer(7 downto 0);
      when x"08" => in_port <= mstimer(15 downto 8);
      when others => in_port <= x"5a";
   end case;
end process readmux;
```

This asynchronously maps the various signals to the picoblaze when it requests them. Note: PORT_ID is valid for 1 extra clock, so it is advisable to put a register in_port, which will allow much better timing *to be reported*.

```
in_port_reg <= in_port when rising_edge(clk);
```

The picoblaze “grabs” data on the rising_edge of clk when read_strobe is high, so if there are any devices like FIFOs for which reads have side effects, this must be taken into account. An example of this is shown on the line below for the uart, which contains an embedded 16-byte fifo, which clearly needs to know when it has been read.

```
read_uart_word <= '1' when port_id = x"01" and read_strobe='1' else '0';
```

Conclusion

The sample design is a perfect illustration of the benefits of the picoblaze processor when combined with custom user logic. In this system, the picoblaze can easily run at a clock rate of 120MHz if necessary, and the design uses only about 2% of the Spartan3E 500 device on the NEXYS-2 board. Hopefully, this design can provide users with a quick-start platform for implementing a picoblaze-based design for their own applications

Questions or problems? Email the instructors:

doug@echelonembedded.com
joseph@echelonembedded.com