Overview of PicoBlaze

PicoBlaze is an efficient 8-bit microcontroller architecture which can be synthesized in Spartan 3 FPGAs (also in Virtex II and Virtex-4). PicoBlaze is similar to many microcontroller architectures but it is specifically designed and optimized for Xilinx FPGAs. There are also larger microprocessors that can be synthesized into FPGAs such as the 32-bit MicroBlaze microprocessor. PicoBlaze and MicroBlaze are typically referred to as soft processor cores since they are synthesized from an HDL and use the programmable logic and routing resources of an FPGA for their implementation, as opposed to a dedicated processor hard core such as the PowerPC that is incorporated in some Virtex II and Virtex-4 FPGAs. However, the process of implementing any soft core microcontroller or microprocessor in an FPGA will be similar to that described here for PicoBlaze. This tutorial assumes that you have downloaded the *PicoBlaze.zip* files from the web page and extracted all of the files into you working directory. These files include everything discussed in this overview and needed to synthesize the example program.

PicoBlaze (see Figure 1) consists of two parts: 1) the processor core (KCPSM3 – which stands for Ken Chapman Programmable State Machine version 3, the person at Xilinx who developed the PicoBlaze) and 2) the program memory from which instructions are fetched and executed by the processor core. (Note that the program memory is referred to as an instruction ROM or program ROM in PicoBlaze documentation since the processor core cannot write to the program memory.) As a result, there are also two VHDL files that are used to construct the complete PicoBlaze with program. The *KCPSM3.vhd* file is optimized for Spartan 3 by calling design primitives specific to Spartan 3 (also for Virtex II and Virtex-4) and, as a result, this VHDL file should not be modified by the user. The KCPSM3 require approximately 96 slices in a Spartan 3. The program memory, on the other hand, is a VHDL file specific to the user's desired program to be executed by the PicoBlaze core and is generated automatically by the assembler (*KCPSM3.exe*) from your assembly language program, *name.psm*. (Note that the prefix for the .psm file must be 8 characters or less.) The program memory is implemented in a single Block RAM in the FPGA configured to function as a 1K×18-bit ROM. The program to be executed is typically initialized in the Block RAM during the download of the overall design (including PicoBlaze and other user logic) and, as a result, is normally assembled prior to synthesis.

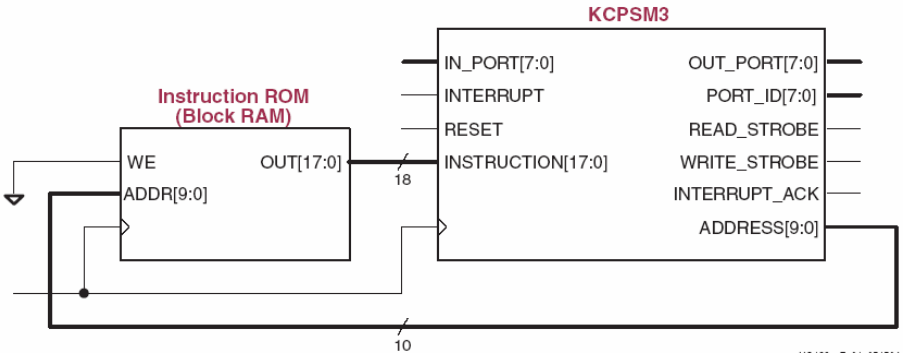

Figure 1. Basic Components of PicoBlaze.

One method of verifying the assembly language program prior synthesis is to use the PicoBlaze simulator, *pBlazIDE.exe*, version 3.6β. Open pBlazeIDE and under **Settings** select **PicoBlaze3**. Under **File** select *Import* and then select your assembly language file that you have written with any text editor. This will import your program and convert some of the PicoBlaze assembly

instructions to simulator specific instructions (for example, the INPUT instruction is converted to IN). As a simple example, consider the following PicoBlaze program which reads 8-bit input data from the IN_PORT, stores it in register s0, reads the data in s0 and writes it to OUT_PORT, and repeats this process by jumping back to the start of the program memory:

```
INPUT s0,00
OUTPUT s0,01
JUMP 000
```

To emulate the input port and output port, we will include the following two pBlazeIDE specific instructions at the beginning of the program which will emulate a set of 8 switches on the input port and a set of 8 LEDs on the output port (with port IDs 00 and 01 respectively).

```
switches  DSIN      $00
LEDS      DSOUT     $01
```

When we import the program (tutorial.psm in this case) into pBlazeIDE and select **Assemble & Simulate**, we obtain a screen similar to that of Figure 2. Input values can be set by the check boxes for the "switches" and output values can be observed on the "LEDs" to the right of the screen. Also note that the contents of the 16 registers (s0-sF) can be observed to the left of the screen along with the status bits and flags. Operation of the simulation is primarily controlled by the 9 buttons shown in the orange oval. From left to right, these buttons control: exit simulation and return to edit mode (blue button), reset the simulation (button with an X), run the simulation (continuously or until a break point is reached) (button with the arrow), single step the simulation (button with a 1), step over the next instruction (button with an S), simulate to cursor, stop or pause the simulation (button with parallel vertical lines), toggle breakpoint (button with hand), and clear all breakpoints.

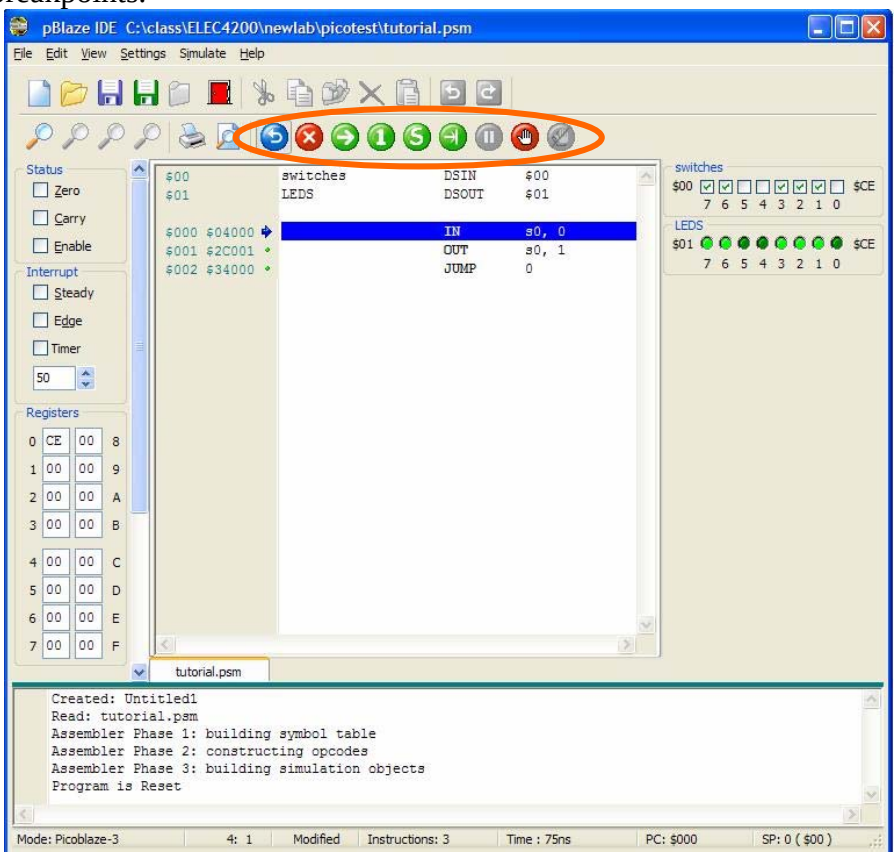

Figure 2. pBlazeIDE Simulator.

Note that the tutorial.psm file in the *PicoBlaze.zip* file includes comment indicators ';' at the beginning of the lines for "switches" and "LEDS" since these are directives to the simulator only and not valid for the assembler. Therefore these comment indicators must be removed for simulation in pBlazeIDE but included for assembly of the program for synthesis and downloading. One technique is to remove the comment indicators in the Edit Mode of pBlazeIDE after importing the file. But do not save the edited file since the format of the saved file will not be compatible with the assembler. In fact, it is recommended that all editing of the assembly language program be done outside of pBlazeIDE in a KCPSM3 compatible format and imported in pBlazeIDE each time a change is made; otherwise, you will need to be familiar with the assembly language syntax differences between pBlaze IDE and the KCPSM3 assembler in order to covert the program saved in pBlazeIDE back to a KCPSM3 compatible format.

Once the program has been simulated and verified, it can be assembled by *KCPSM3.exe*. First we will need to comment out any pBlazeIDE specific instructions (like the "switches" and "LEDS") by inserting a semicolon at the beginning of each line. Next, open a command line tool and type the following command to assemble the program and generation the program memory VHDL file:

```
KCPSM3 tutorial.psm
```

This will produce a file named *tutorial.vhd* which contains the VHDL for a Block RAM instantiation which is initialized with the machine language for the assembled program. Note that you will need to have the files *ROM_form.vhd* and *ROM_form.coe* in your directory where you are compiling the program. Now you will need a top level VHDL model to instantiate and interconnect the PicoBlaze core (*kcpsm3.vhd*) and the program memory (*tutorial.vhd*). The example given in the *PicoBlaze.zip* file is called *toptut.vhd* and has comments to assist in understanding the instantiation and interconnection of the VHDL models to construct the complete microcontroller for this example. Next you can create a new project in ISE and add in the *toptut.vhd, kcpsm3.vhd,* and *tutorial.vhd* files. Create a constraints file to connect the clock input to the oscillator on the Spartan 3 board and to connect the switches and LEDs on the board to the input and outport of the PicoBlaze. The remainder of the synthesis and downloading process is the same as in previous labs.

PicoBlaze and the KCPSM3 assembler support a total of 57 instructions, each belonging to one of 7 groups. There are approximately 21 types of instructions and, hence, the instruction set is fairly small and easy to learn. Instructions can be upper or lower case characters but are converted by the assembler to upper case (see the .fmt file produced by the assembler). Comments begin with a semicolon, and everything on the line following the semicolon is ignored by the assembler. Note that the KCPSM3.vhd is compatible with the ModelSim simulator and can be simulated with the assembled program memory and any other user designed logic. More information on the PicoBlaze and the KCPSM3 assembler, including the pBlazeIDE simulator and VHDL simulation in ModelSim can be found in the "PicoBlaze 8-Bit Microcontroller User Guide" and in the "KCPSM3 Manual". Both are available on the class web page (www.eng.auburn.edu/~strouce then click on the link for ELEC 4200 Digital System Design).

Happy PICOing!!!