References:
PicoBlaze manual
http://www.xilinx.com/support/documentation/user_guides/ug129.pdf
PicoBlaze: http://www.xilinx.com/products/ipcenter/PicoBlaze-S3-V2-Pro.htm
VHDL template: http://www.mediatronix.com/code/ROM_blank.vhd
pBlazeIDE:http://www.mediatronix.com/pBlazeIDE.htm

SETUP

1. Download PicoBlaze (Registration required): http://www.xilinx.com/products/ipcenter/PicoBlaze-S3-V2-Pro.htm
2. Download the PicoBlaze IDE: http://www.mediatronix.com/pBlazeIDE.htm
3. Download the VHDL template from here http://www.mediatronix.com/code/ROM_blank.vhd
4. Extract PicoBlaze and the PicoBlaze IDE.
5. Add all of the PicoBlaze files to your project in Xilinx.
6. Run PicoBlaze IDE (requires no installation)
7. Write assembly program. (see below for details)
8. Assemble the program, this will generate a VHD file.
9. Add the new VHD file to the project in Xilinx.
10. Link the new VHDL module created by the PicoBlaze IDE into the VHDL template
11. Build and download the VHDL to the FPGA.

To write a PicoBlaze program, start with the code template, which can be found in the PicoBlaze manual, then use the PicoBlaze manual as a reference for the assembly language that is used. The first thing to note in the code template is the line that begins with VHDL, this line specifies the name of the VHDL template that will be used when assembling (use the template downloaded from Mediatonix as a base), the name of the output VHD file which the assembler will create, and the name of the module that will contain the assembled code.

In order to interface with the VHDL setup running outside of PicoBlaze, simply declare a new port, using the statement “name DSIN port” or “name DSOUT port” where name is the name you want to assign to the port, and port is a number representing what port you are creating. On the VHDL side of the interface, there is a single signal for in-put and a single signal for output, so to gain access to the different ports you create a multiplexer based on the port numbers assigned in the assembly code. The IDE includes in it a compiler and a simulator, use the simulator to thoroughly test the assembly code, as it shows the values of all available registers as well as all input and output ports. This is the best way to debug the assembly code because once you get the code running in the actual FPGA there is limited access to information for debugging.

To add the PicoBlaze processor into a VHDL file, components must be declared for both the processor and the program. To declare components, copy the following code into the VHDL file, in the same entity declaration where signals are declared (before “begin”). The signals are not necessary, but useful

```
      component <program entity name, found in output from assembler>
      port (
            address : in std_logic_vector( 9 downto 0);
            instruction : out std_logic_vector(17 downto 0);
            clk : in std_logic
      );
      end component;

      component KCPSM3
      port (
            address : out std_logic_vector( 9 downto 0);
            instruction : in std_logic_vector(17 downto 0);
            port_id : out std_logic_vector( 7 downto 0);
            write_strobe : out std_logic;
            out_port : out std_logic_vector( 7 downto 0);
            read_strobe : out std_logic;
            in_port : in std_logic_vector( 7 downto 0);
            interrupt : in std_logic;
            interrupt_ack : out std_logic;
            reset : in std_logic;
            clk : in std_logic
      );

      signal instruction_signal     : std_Logic_vector(17 downto 0);
      signal address_signal         : std_logic_vector(9 downto 0);
      signal port_id_signal         : std_logic_vector(7 downto 0);
      signal out_port_signal        : std_logic_vector(7 downto 0);
      signal in_port_signal         : std_logic_vector(7 downto 0);
      signal write_strobe_signal    : std_logic;
      signal read_strobe_signal     : std_logic;
      signal interrupt_signal       : std_logic;
      signal interrupt_ack_signal   : std_logic;
      signal reset_signal           : std_logic;
      signal clk_signal             : std_logic;
```

Next, instances of these components must be added into the VHDL behavioral section. The following code creates one of each component and "wires" it to the signals declared above.

```
      processor: kcpsm3
            port map(
            address => address_signal,
            instruction => instruction_signal,
            port_id => port_id_signal,
            write_strobe => write_strobe_signal,
            out_port => out_port_signal,
            read_strobe => read_strobe_signal,
            in_port => in_port_signal,
            interrupt => interrupt_signal,
            interrupt_ack => interrupt_ack_signal,
            reset => reset_signal,
            clk => clk_signal
```

```
);

program: proofProgram
      port map( address => address_signal,
      instruction => instruction_signal,
      clk => clk_signal
);
```

To use these components, simply connect the signals to other components or input and output switches. For simple programs (with only one input and one output port for the processor), no multiplexer is needed, and the port_id can be ignored. A simple program, good for testing the PicoBlaze functionality, is connecting the in_port_signal directly to the FPGA board's switches, and the output port to the LEDs. Write some simple program for the PicoBlaze that continuously reads the inputs and writes to the outputs to verify that PicoBlaze is being used properly.