

How-To User Guide: Working with PicoBlaze on an FPGA

By: Sean Farner
Peter Frandina

Date: 5/20/2008

Table of Contents

1.0 Background Information.....	3
2.0 File Setup.....	3
3.0 Editing Source Files	3
4.0 Compiling Source Files	4
5.0 VHDL Implementation of the PicoBlaze Components	5
6.0 Communicating Between Assembly and VHDL	6
7.0 Reference Material	7

1.0 Background Information

PicoBlaze offers an 8-bit micro controller for Spartan-3 FPGAs (Other versions of PicoBlaze are available for other devices). This micro controller occupies just 96 Spartan-3 slices and utilizes a single block RAM to form a ROM store for a program of up to 1024 instructions. The figure below shows a block diagram of the micro controller connected to the program block of ROM.

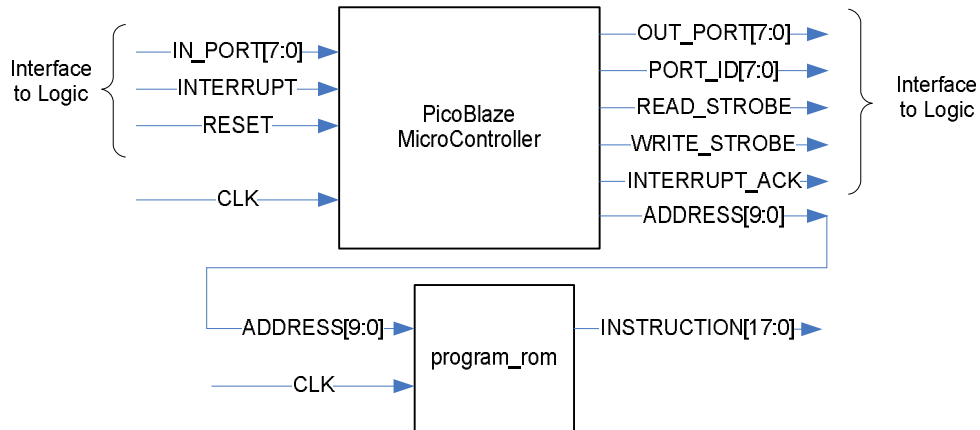


Figure 1 - Microcontroller Connected to Block ROM

2.0 File Setup

Make sure that the following files are in the PicoBlaze development directory:

- | | |
|--------------|---|
| KCPSM3.EXE | - Compiler |
| progame.psm | - User's assembly source file |
| ROM_form.coe | - used to define the header information |
| ROM_form.vhd | - used to create a vhdl file |
| ROM_form.v | - used to create a verilog file |

3.0 Editing Source Files

A basic text editor (Vim/Notepad) can be used to create and edit the assembly source file to run on the PicoBlaze. Only the provided set of instructions that are listed in the PicoBlaze user manual should be used. The source file should be saved with a .psm extension. NOTE: The compiler will not accept filenames longer than 8 characters.

4.0 Compiling Source Files

After writing a source file, it can be compiled using the KCPSM3 executable. From a command prompt in the PicoBlaze development directory, compile the assembly source file by invoking the following command:

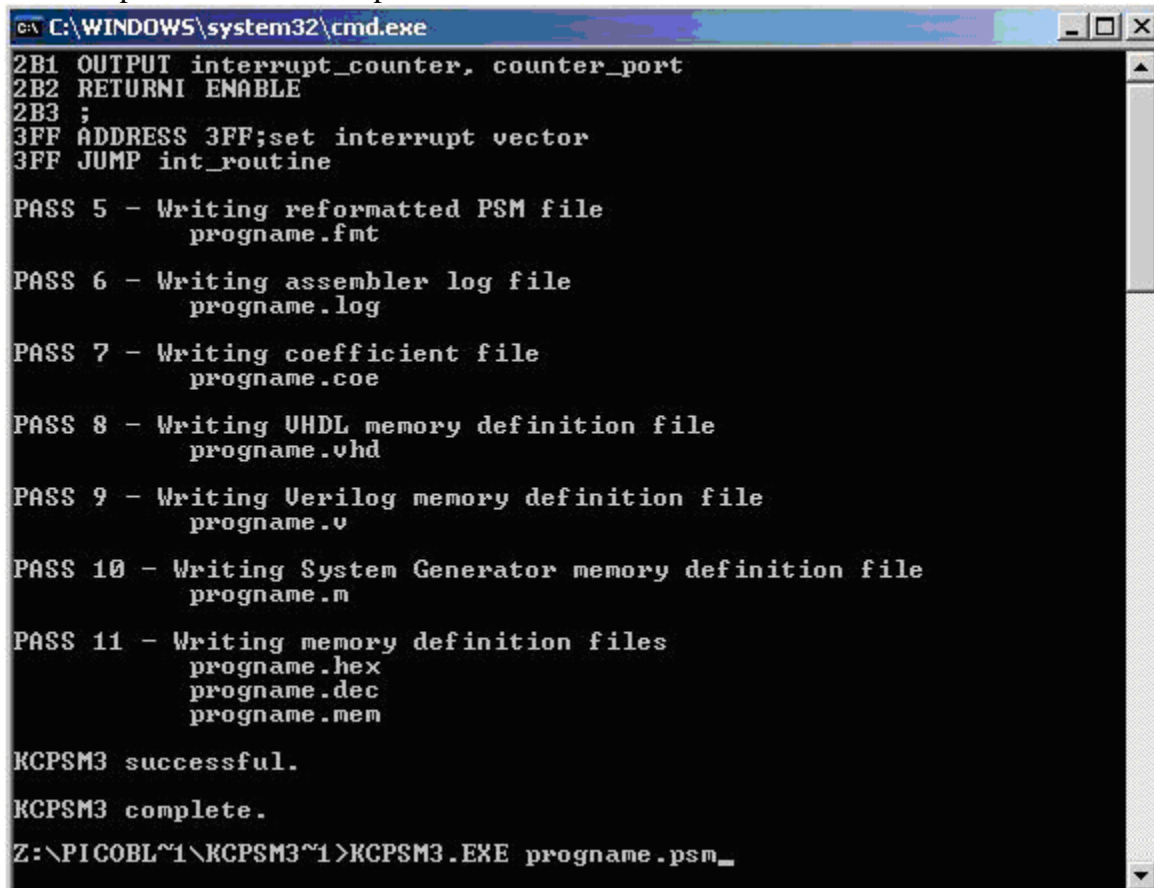
```
KCPSM>KCPSM3.EXE progame.psm
```

This executable will run through several tests and notify you if any errors exist in the .psm file. It creates several files including the following files of importance:

```
progame.vhd - VHDL version of the assembly program
progame.v - Verilog version of the assembly program
```

For a VHDL project, copy the progame.vhd file that was created by the compiler into the FPGA development folder. This file should be included in the FPGA project.

An example view of the compiler is shown below:



```
C:\WINDOWS\system32\cmd.exe
2B1 OUTPUT interrupt_counter, counter_port
2B2 RETURNI ENABLE
2B3 ;
3FF ADDRESS 3FF;set interrupt vector
3FF JUMP int_routine

PASS 5 - Writing reformatted PSM file
        progame.fmt

PASS 6 - Writing assembler log file
        progame.log

PASS 7 - Writing coefficient file
        progame.coe

PASS 8 - Writing VHDL memory definition file
        progame.vhd

PASS 9 - Writing Verilog memory definition file
        progame.v

PASS 10 - Writing System Generator memory definition file
        progame.n

PASS 11 - Writing memory definition files
         progame.hex
         progame.dec
         progame.mem

KCPSM3 successful.
KCPSM3 complete.
Z:\PICOBL~1\KCPSM3~1>KCPSM3.EXE progame.psm_
```

Figure 2 - Compiler Output at Command Prompt

5.0 VHDL Implementation of the PicoBlaze Components

In order to develop synthesizable code for the FPGA, the provided `kcpsm3.vhd` must be added to the project along with the PicoBlaze generated `progame.vhd` file. The figure below shows how PicoBlaze is wired into the generated VHDL. The generated code is addressed by PicoBlaze and executes instructions at the rate of the clock.

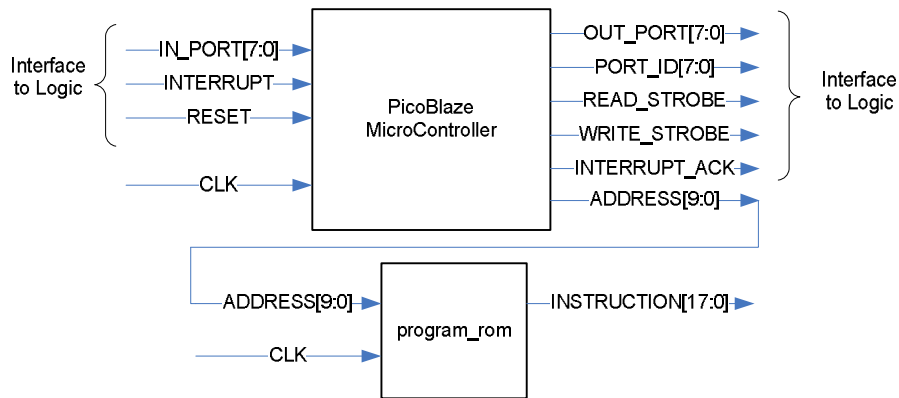


Figure 3 - PicoBlaze MicroController wired into the Assembly Generated VHDL

6.0 Communicating Between Assembly and VHDL

Communication is essentially achieved by issuing INPUT and OUTPUT commands in the assembly program. Both of these commands require a value to be included for the port_id. In the VHDL code, a mux can be used to determine what the assembly program is trying to communicate. One mux is used for the INPUT command and another for the OUTPUT command.

The INPUT mux will be enabled by the READ_STROBE signal from PicoBlaze. This strobe indicates that an INPUT command was issued. The mux can then use the specified port_id to decide which data should be sent to the MicroController. An example of this INPUT mux is seen below.

```
input_ports: process(clk)
begin
  if rising_edge(clk) then
    case port_id is
      when x"00" => -- read UART status at address 00 hex
                    in_port <= uart_status_port;
      when x"01" => --read status byte
                    in_port <= status_port;
      when others => --Don't care used for other addresses
                    in_port <= "XXXXXXXX";
    end case;
  end if;
end process;
```

The OUTPUT mux is similar to the INPUT mux, except it is enabled by the WRITE_STROBE signal from PicoBlaze. When this signal is active, it signifies that the assembly code issued an OUTPUT command in an attempt to send data from the Microcontroller to the VHDL logic. The port_id can be used to determine which data should be expected and which hardware logic to use. An example of this type of mux is seen below.

```
output_ports: process(clk)
begin
  if rising_edge(clk) then
    if write_strobe='1' then
      case port_id is
        when x"00" => -- Turn on/off led(0) at address 00 hex
                      led(0) <= out_data;
        when x"01" => -- Write to UART at address 01 hex
                      write_to_uart <= '1';
        when others => --Do nothing to prevent errors
                      null;
      end case;
    end if;
  end if;
end process;
```

7.0 Reference Material

The following link is to the Xilinx website, where the PicoBlaze files and documentation are available for download.

<http://www.xilinx.com/products/ipcenter/picoblaze-S3-V2-Pro.htm>