Guide to using and extending the PacoBlazeSM project

Table of Contents

[Background 1](#_Toc309685357)

[Needed tools 2](#_Toc309685358)

[Reference Material 2](#_Toc309685359)

[Project setup and configuration 2](#_Toc309685360)

[Simulation 3](#_Toc309685361)

[Running the simulation 3](#_Toc309685362)

[Synthesis 3](#_Toc309685363)

[Running the Synthesis process 4](#_Toc309685364)

[Debugging 4](#_Toc309685365)

[Breakpoints and Watches 4](#_Toc309685366)

[Waveform viewer 5](#_Toc309685367)

[Working with the KCASM assembler 5](#_Toc309685368)

[Add java jdk to path (if it’s not already set) 5](#_Toc309685369)

[Run kcasm assembler on an assembly (psm) file 5](#_Toc309685370)

[Run kcasm assembler on a group of assembly (psm) files from the cmd line 5](#_Toc309685371)

[Modifying the KCASM assembler 6](#_Toc309685372)

[Creating a java project for KCASM assembler in Eclipse 6](#_Toc309685373)

[Modifying the PacoBlazeSM project 6](#_Toc309685374)

[Modifying number of CPU’s and Butterfly Network Size 6](#_Toc309685375)

[Modifying System wide macros that control various aspects of the PacoBlazeSM processor 7](#_Toc309685376)

# Background

PacoblazeSM is a 32-bit multi core CPU system that uses a butterfly shared memory model.

The hardware part of the system is written in Verilog. It is based on an open source 8-bit CPU called Pacoblaze which is a PicoBlaze open source clone.

The original processor was extended to 32-bit and was modified to support the new shared memory model.

The software part of the system, i.e. the assembler for the processor is written in Java.

# Needed tools

* XILINX ISE Web pack (free edition) –

 <http://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.htm>

 \* Project was tested on version 12.3 and up.

* Java JDK + Eclipse -

You can download the JDK from:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

\* Eclipse is recommended if you want to easily change the PacoBlaze assembler project. It is not a prerequisite.

# Reference Material

PicoBlaze 8-bit Embedded Microcontroller User Guide

<http://www.xilinx.com/support/documentation/ipembedprocess_processorcore_picoblaze_.htm>

\* Useful information regarding the original PicoBlaze processor including the instruction set and instruction structure.

# Project setup and configuration

1. Download the source code and Projects from:

<http://cs.haifa.ac.il/courses/verilog/emc.zip>

2. Extract to a folder located on your computer where you want the project root to reside.

 \* There is currently a zip inside a zip so extract twice in the same location

 \* The project size after simulation and synthesis can reach more than 100MB so make sure you have enough space on the disk.

3. using your file explorer go inside the folder **[Project Location]\PacoblazeSM** and double click the project file called **PacoblazeSM.xise**

This should automatically open the project inside Xilinx ISE.

# Simulation

- Always remember to change to simulation mode, choose a test bench and then rerun all.

 \* If you forget to choose a test bench file it will either fail or will only run the component you chose. i.e. not all the system which is simulated by the test bench.

- Use the 4X test bench for your initial tests (pacoblazeSM\_4XCPU\_tb.v). It is a 4XCPU+4XBFNet simulation.

It is tested and working on the assembly files for matrix multiplication.

## Running the simulation



# Synthesis

- Always remember to change to synthesis mode, choose a top module and then rerun all.

 \* If you forget to choose the correct top module it will either fail or will only run the component you chose. i.e. not all the system which want to synthesize.

- Use pacoblazeSMSys as your top module. It contains the complete PacoblazeSM System.

## Running the Synthesis process



# Debugging

There are several elements that make programming in a hardware definition language (HDL) like verilog, difficult when compared to procedural programming languages:

1. The element of Time – operations take time to complete.

2. Parallel programming gone wild – different parts of the system are changing continually without the synchronizations primitives we are used to rely on (Mutexes ,Semaphors etc.).

For those reasons debugging your hardware modules is invaluable in the development and testing stages and the Xilinx ISE offers several useful features to help to do so.

## Breakpoints and Watches

* You can set breakpoints at different parts of your code. When a breakpoint is hit you can check the values of various objects in the system.
* You can also watch memory structures and examine their values when a program is in stasis.
* You can run a simulation for a specific duration of time after which the simulation will pause and you can examine different objects in the system.

## Waveform viewer

* The waveform can become your best friend or your worst enemy but you will have to learn to use it.
* The waveform viewer lets you view the signals propagating in the system during the simulation.
* You can see what happens in the system at any given time of the simulation. When you break/pause the simulation you can examine the waveform as well.

# Working with the KCASM assembler

The KCASM assembler transforms the assembly files to a form that can be used in the simulator and in synthesis. Originally it supported the 8-bit PacoBlaze so it was modified to support the new 32-bit version.

KCASM produces 2 types of files:

Memory Hex Files (\*.rmh) – Pacoblaze machine code files in hexadecimal format, that can easily be loaded by a custom loader or in simulations using the verilog read memory command (readmemh).

Verilog files (\*.v) – Synthesizable Pacoblaze machine code files written in Verilog (was used for Synthesis in the original version).

\* The RMH files are the ones that are currently used in the simulations.

## Add java jdk to path (if it’s not already set)

SET PATH=%PATH%;%JAVA\_HOME%\bin

## Run kcasm assembler on an assembly (psm) file

1. Open cmd line

2. Go to [project location]\pacoblazeFiles\util

3. java -cp ..\kcasm\bin KCAsm kcpsm=3 bram=18 module=cpu\_0\_code\_mod asmFile="..\test\cpu\_0\_code.psm" rmhFile="..\test\cpu\_0\_code.rmh"

## Run kcasm assembler on a group of assembly (psm) files from the cmd line

1. Open cmd line

2. Go to [project location]\pacoblazeFiles\util

3. Run compile\_asm\_code.bat

# Modifying the KCASM assembler

## Creating a java project for KCASM assembler in Eclipse

- Create a new workspace (or use an existing one)

- Import new project -> file/import/general/existing projects

 Ignore import error message if you have one...

- Examples of KCASM assembler cmd line params ->

kcpsm=3 bram=18 module=pbsm\_test\_mod asmFile="..\test\cpu\_0\_code.psm" rmhFile="..\test\cpu\_0\_code.rmh"

kcpsm=3 bram=18 module=pbsm\_test\_mod asmFile="..\test\pb3m\_sm\_test.psm" rmhFile="..\test\pb3m\_sm\_test.rmh"

# Modifying the PacoBlazeSM project

## Modifying number of CPU’s and Butterfly Network Size

There are 2 macros that control the size of the system BFNETWORK\_SIZE and NUM\_OF\_CORES

Example:

To create a 4 CPU’s and 4 X BF Network we would set the macros under the Implementation Process properties/Verilog Macros in the following way:

BFNETWORK\_SIZE=4 | NUM\_OF\_CORES=4

\* Note that BF net size and num of cores should always be the same. The 2 macros control 2 different sub projects.



## Modifying System wide macros that control various aspects of the PacoBlazeSM processor

The file **PacoblazeSM\pacoblazeFiles\pacoblaze\pacoblaze\_inc.v** holds many of the system parameters.

Some examples:

- The macro DATA\_WIDTH\_BITS sets the system data width (defined as 32 bit for now.)

- The macro CODE\_WIDTH\_BITS controls the instruction size (opcode size + reg size + ram addr size)