PicoBlaze™

8-bit Pipelined Picoblaze

NIKO NIKOLOV, AALAP KHANOLKAR, KATHYAYANI NEERUDI

XILINX®

Overview

- Introduction
- Block Diagram
- Project Goals
- Verification approach
- Pipeline stages
  - Instruction fetch
  - Instruction decode
  - Execute
  - Memory access
  - Write back
- Assertions
- Bugs found
- Challenges faced
- Learnings
- References

Introduction

- Picoblaze is a Xilinx microcontroller
- Used in many Xilinx FPGAs such as Series 7
- Pacoblaze is RTL implementation of the Picoblaze ISA
- Pipelined Picoblaze (RojoBlaze) is based on the Pacoblaze
- Rojoblaze written by John Lynch and Roy Kravitz

Picoblaze Block Diagram

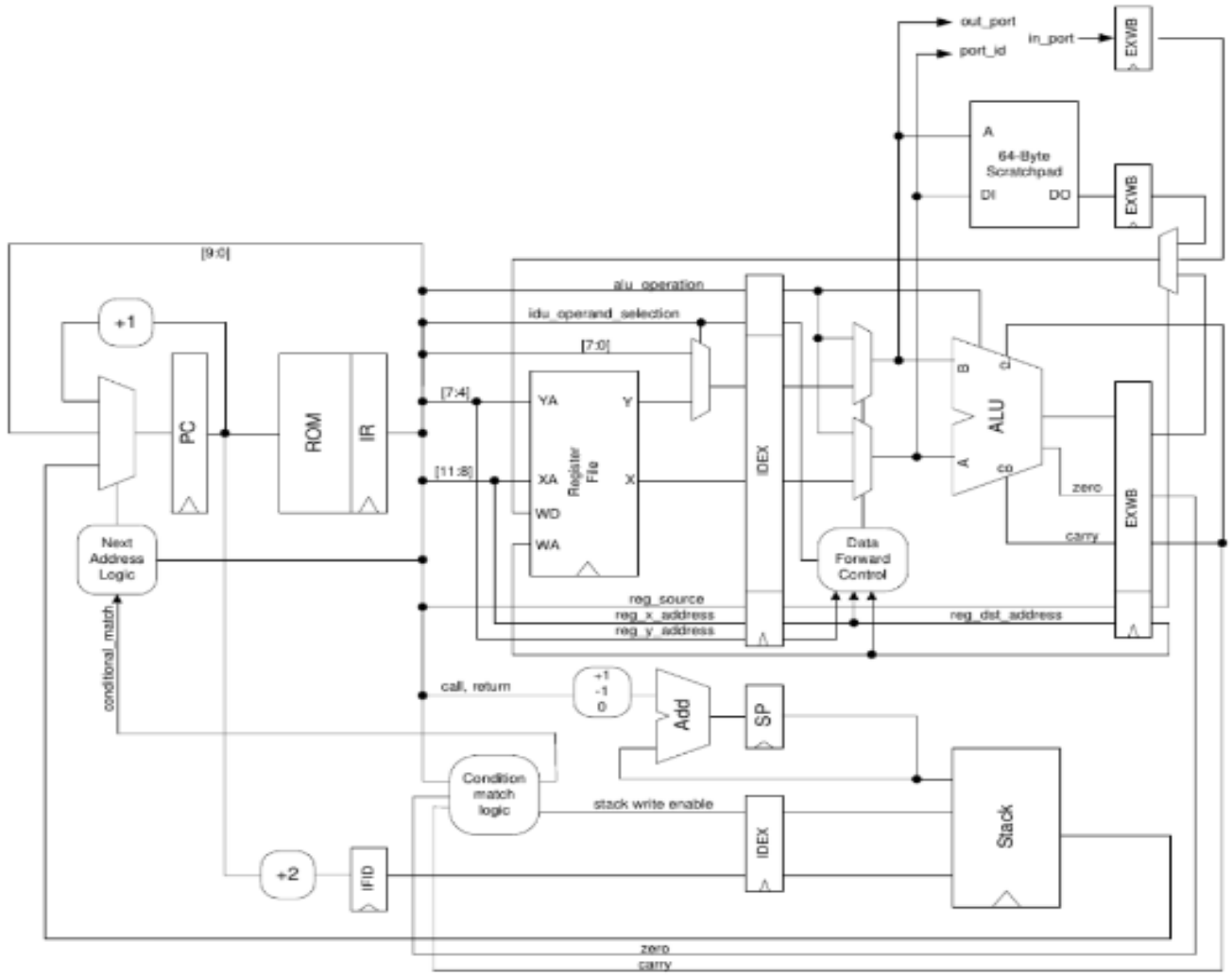


Figure: Picoblaze block diagram

Project Goals

- Verify the pipelined Picoblaze RTL model
- Understand Micro Controller functionality
- Breakdown testing to different segments .i.e. Instructions and signal propagation
  - ►Test instructions behavior
  - ►Test signals propagation and timing
- Writing test cases and assertions to find the bug in the code.

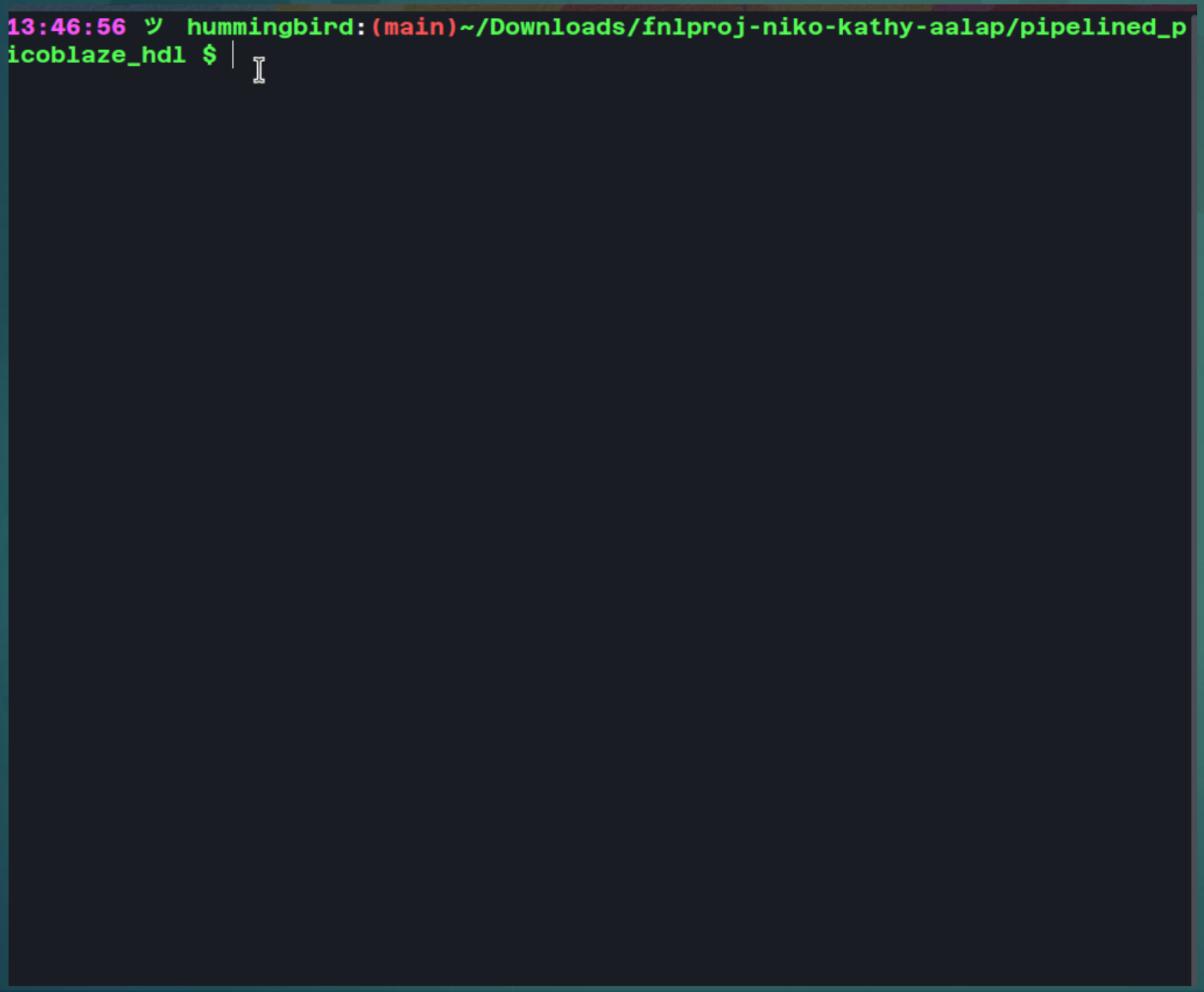
Verification Approach

- Assembly code (.psm files) for the Picoblaze ISA
- Unit tests for 5 pipeline stages (IF, ID, EX, MA, WB)
- Generate .mem files using the Picoblaze assembler
- Instantiate and run .mem files in the top level testbench as stimulus to the Picoblaze
- Assertions have been used to text the bugs on every stage of the pipeline.
- Every assertion is compared with the bugged and non-bugged (gold standard) code.
- If a particular assertion is triggered

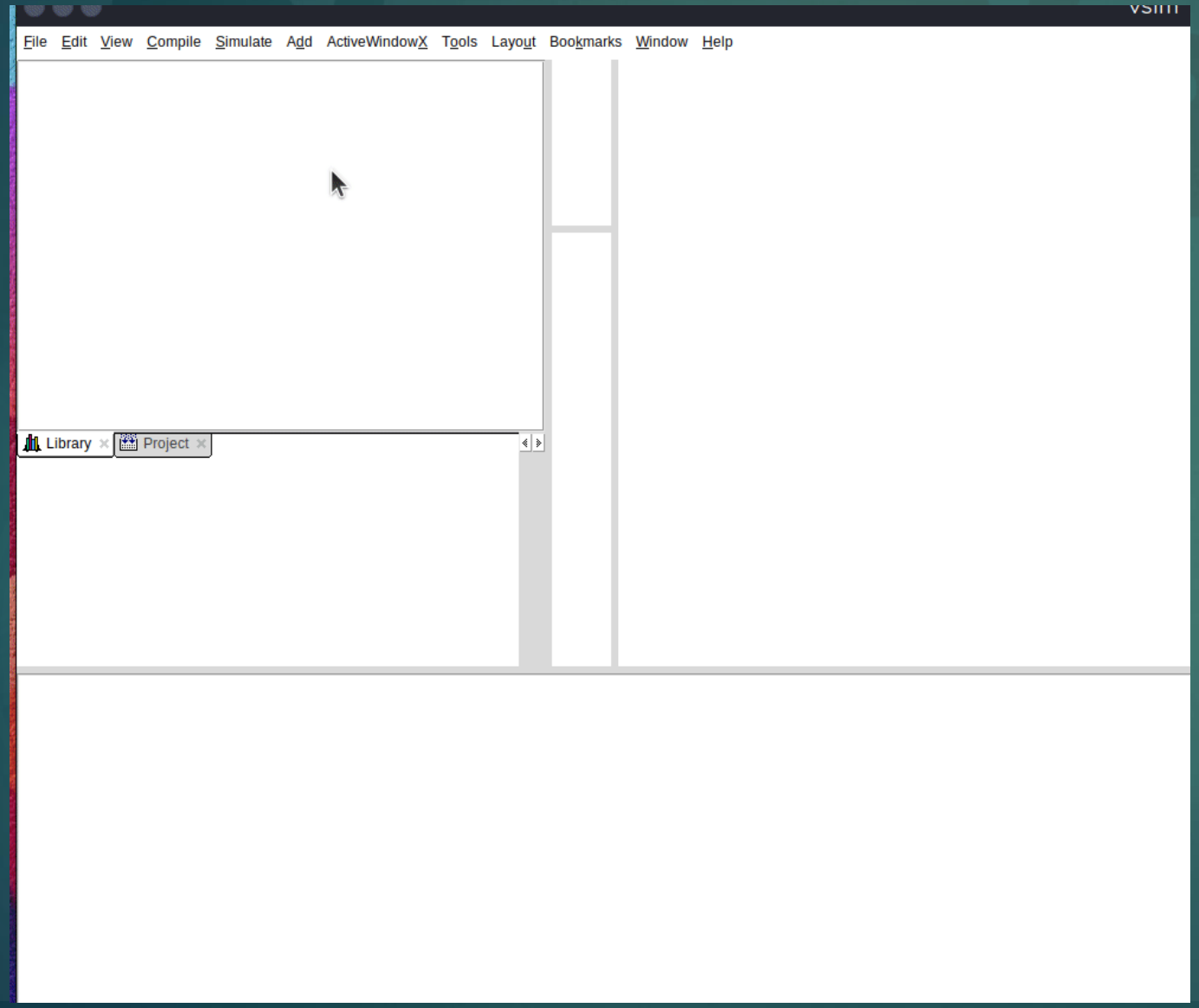
Using KCAsm


```
14:05:22 ツ  hummingbird:(main)~/Downloads/fnlproj-niko-kathy-aalap/pipelined_picoblaze_samp
le_programs $
```

Compiling PicoBlaze psm



Simulating mem file in testbench



Demonstrating assertions in different pipeline stages

This architecture consists of 5 stages, namely-

- Instruction Fetch
- Instruction Decode
- Instruction Execute
- Memory Access
- Write Back

Instruction Fetch

The instruction fetch unit fetches the instruction that is currently stored in the program counter. The assertions that we wrote are:

- reset_check: checks if PC = zero when reset signal is asserted
- rollover_check: Checks if PC rollover is working
- pc_jump: Checks if PC is jumping to the target address

Instruction Decode

The decoding process allows the CPU to determine what instruction is to be performed so that the CPU can tell how many operands it needs to fetch in order to perform the instruction. The assertions that we wrote were as follows:

- Testing the validity of the instructions (Error if the instruction is invalid)
- Checking the ZERO flag
- Checking the CARRY flag.

Instruction Execution

- Validate ALU operations such as ADD and SUB
- Validate the ZERO flag is asserted correctly
- Validate the CARRY flag is asserted correctly
- Validate the Carry Flag is cleared correctly
- Validate the shift_bit is asserted correctly
- Validate write and read strobe
- Validate scratch write enable
- Validate register write enable
- Validate ALU operand_a and operand_b are correctly asserted

Memory Access

In this segment the primary thing that we want to test is whether we are accessing the memory in the correctly.

The assertions that we wrote to test the following were:

1. To ensure correct memory access, check we set an assertion to check the Scratchpad boundary
2. Check Scratch Write gets asserted after STORE
3. Check Register Write gets asserted during STORE
4. Check WR Strobe gets asserted when writing to Scratch Register

Memory Writeback

1) Validate in_port contains the correct value before writing
2) Validate reg_source contains is asserted correctly to multiplex the WB
3) Validate dist contains the correct register value
4) Validate X and Y addresses are populated correctly
5) Validate X and Y match operand_a and operand_b
6) Validate if STORE bits [3:0] are zero

Bugs Found

- ALU not setting the Zero Flag
- ALU not performing correctly ADD and SUB
- ALU not setting the Carry Flag correctly
- STORE not storing the data in the scratchpad due to decrementing incorrectly and ALU wrong Address calculation (later we were explained it was due to the stack pointer)

Challenges Encountered

1. PicoBlaze architecture and its functioning was difficult to understand.
2. Eliminating false positives from with the assertions was also a crucial step.
3. Coordinating remotely
4. Given the limited amount of time it was difficult to decide what we should test first
5. CAT infrastructure goes down or slows down often

Learnings

1. We got to work collectively in the github classroom version control environment.
2. Working together remotely, exchanging ideas and collaborating helped all three of us to get better insights of the PicoBlaze, writing the test cases and assertions.
3. Timing is literally everything when it comes to digital design and testing

Given more time, what we would do differently...

1. Review more functionalities and write higher level tests/assertions
2. Drill down to timings and signal level testing, to ensure signals propagate based on the spec defined timing and get asserted correctly
3. Write tests for stack pointer
4. Instead of having the assertions in the test bench we would have created a separate file for that, even better separate the assertions
5. We would have created an interface that connects all signals, instead of hacking the signals through their module names. That would have been a much cleaner approach

References Used

1. Sutherland, S. (2017). *RTL Modeling with SystemVerilog for Simulation and Synthesis using SystemVerilog for ASIC and FPGA design*. Sutherland HDL, Inc.
2. Mehta, A. (2016). *SystemVerilog Assertions and Functional Coverage Guide to Language, Methodology and Applications*. Springer International Publishing AG Switzerland.
3. Taraate, V. (2020). *SystemVerilog for Hardware Description RTL Design and Verification*. Springer.
4. Chapman, K. (2003). Picoblaze KCPSM3 8-bit Micro Controller for Spartan-3, Virtex-II and Virtex-IIPRO. Xilinx Ltd.
5. Herriman, A., & Blanchard, E. (2006). PacoBlaze. Xilinx Ltd.
6. Chapman, K. (2005). PicoBlaze 8-bit Embedded Microcontroller User Guide for Spartan-3, Virtex-II, and Virtex-II Pro FPGAs. Xilinx.