INSTRUCTION COMMIT-LEVEL PC TRACING IN SHAKTI C-CLASS VIA BRAM

SUMMER INTERNSHIP - I REPORT

Submitted by

ESHAANJANA S - 2023105011

TANUJA SREE R - 2023105013

in partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

in

ELECTRONICS AND COMMUNICATION ENGINEERING

COLLEGE OF ENGINEERING, GUINDY

ANNA UNIVERSITY: CHENNAI 600025

MAY - JULY 2025

INSTRUCTION COMMIT-LEVEL PC TRACING IN SHAKTI C-CLASS VIA BRAM

SUMMER INTERNSHIP – I REPORT

Submitted by

ESHAANJANA S – 2023105011

TANUJA SREE R- 2023105013

in partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

in

ELECTRONICS AND COMMUNICATION ENGINEERING

COLLEGE OF ENGINEERING, GUINDY

ANNA UNIVERSITY: CHENNAI 600025

MAY - JULY 2025

ANNA UNIVERSITY: CHENNAI 600025

BONAFIDE CERTIFICATE

Certified that this report titled as, **"INSTRUCTION COMMIT-LEVEL PC TRACING IN SHAKTI C-CLASS VIA BRAM "** is the Bonafide work of **ESHAANJANA S (2023105011) & TANUJA SREE R (2023105013)** for 5th Semester who carried out the project work for Summer Internship-I, in the period of May - July 2025 under my supervision. Certified further that to the best of my knowledge, the work reported herein does not form part of any other thesis or dissertation based on which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE **SIGNATURE**

Dr. Nitya Ranganathan **Mr. Sriram**

ACKNOWLEDGEMENT

We express our sincere gratitude to the Dean, **Dr.K.S.Easwarakumar**, Professor, College of Engineering, Guindy for his support throughout the project.

We extend our heartfelt appreciation to our Head of the Department, **Dr.M.A Bhagyaveni,** Professor, Department of Electronics & Communication Engineering for her enthusiastic support and guidance throughout the project.

We are immensely grateful to our project supervisors **Dr. Nitya Ranganathan and Mr. Sriram from the SHAKTI Team**, for their unwavering assistance, patience, valuable guidance, technical mentorship, and encouragement in our project.

Certificate page

ABSTRACT

This project implements instruction-level Program Counter (PC) tracing in the Shakti C-Class RISC-V core to enhance RTL debug visibility. The PC is captured at every instruction commit from the write-back stage and stored sequentially into a dedicated BRAM. To make this data accessible from the software side, we instantiate a separate memory-mapped BRAM module in the top-level SoC, connected through the AXI bus, allowing GDB to inspect the trace via address-mapped I/O.

We use a three-window debug setup involving Verilator (for RTL simulation), OpenOCD (for debug bridging), and GDB (for memory inspection) to validate the implementation. Modifications were made to several modules including stage5.bsv, Soc.defines and top-level integration files like ccore.bsv, riscv.bsv, and Soc.bsv.

The setup was tested using RISC-V assembly programs, successfully verifying that each committed instruction's PC is accurately logged and viewable from the host debugger. This enhances real-time observability of processor execution and provides a foundation for deeper debug and trace features in RISC-V systems.

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

1.1 PROJECT MOTIVATION AND OVERVIEW

Modern processor design demands robust debugging and observability tools. In RISC-V, having visibility into the internal pipeline activity such as the Program Counter (PC) at the point of instruction commit is crucial. This project is motivated by the need to bridge the RTL-level simulation environment with higher-level debugging tools like GDB. The work revolves around enabling real-time PC tracing during instruction commits in the Shakti C-Class core and making the trace data accessible via memory-mapped BRAM for inspection.

1.2 PROBLEM STATEMENT

C-Class core/trace/log RTL implementation, simulation and FPGA: Add support to send and read the PC instructions from instruction commit log (rtl.dump). To test on coremark with gdb in simulation and FPGA.

1.3 OBJECTIVES

- Tap the Program Counter (PC) from the write-back stage of the Shakti pipeline.
- Log each committed PC value into a BRAM buffer.
- Instantiate the BRAM as memory-mapped I/O in the SoC.
- Make the BRAM data accessible via GDB using OpenOCD.
- Validate the implementation using RISC-V test programs and CoreMark.

CHAPTER 2

SETUP AND SYSTEM DESIGN

2.1 SHAKTI C-CLASS 4.6.0

The SHAKTI C-Class is a 64-bit in-order, five-stage pipelined RISC-V core developed at IIT Madras as part of the SHAKTI processor initiative. Version 4.6.0 of the C-Class core introduces stable simulation and debugging support, making it ideal for research and development projects. It supports the RV64IMAFDC instruction set extensions and features AXI4 interfaces for memory and peripheral communication. The core is implemented in Bluespec SystemVerilog (BSV), providing high-level modularity and ease of customization. This version also supports integration with Verilator, OpenOCD, and GDB, which is crucial for implementing and validating the PC trace mechanism designed in this project.

2.2 3 - WINDOW SETUP

This setup allows real-time inspection of PC trace data as the program runs in simulation.

- **Verilator**: Simulates the Shakti C-Class SoC with modified modules.
- **OpenOCD**: Acts as a bridge between Verilator and GDB.
- **GDB**: Used to inspect the BRAM content through memory-mapped access

.

2.3 ROLE OF WRITEBACK STAGE

The write-back stage is chosen for PC logging because it represents the final commit point of an instruction. At this stage, the instruction has completed execution and any updates to architectural state have occurred, ensuring a valid PC is captured.

2.4 PC LOGGING MECHANISM

The PC logging mechanism works by adding a FIFO (pc_trace_fifo) in Stage5.bsv that stores the PC of each committed instruction in all commit rules, exposing it through a new pc_trace_out interface. In riscv.bsv, this interface is propagated upward so that ccore.bsv can access it. Inside ccore.bsv, a new rule rl_write_pc_trace dequeues PCs from pc_trace_out and writes them sequentially to a memory-mapped region (0xA0000000 + trace_addr) via the AXI master interface. In Soc.bsv, a new BRAM (trace_bram) is instantiated at base address 0xA0000000 and connected to the AXI fabric as a new slave (with Trace_slave_num defined in Soc.defines). The AXI fabric's Num_Slaves was increased to include this new BRAM. As a result, during simulation, every committed instruction's PC is logged sequentially into this BRAM, and since the BRAM is memory-mapped, GDB or software can read the PCs directly from this address region to trace execution.

CHAPTER 3

IMPLEMENTATION

3.1 PC TRACING AND BRAM INSTANTIATION

In order to trace the Program Counter (PC) of every committed instruction in the Shakti C-Class core, a dedicated mechanism was implemented in the writeback stage (Stage5) and extended across the processor system. This section describes how PC values are collected and stored in a memory-mapped BRAM for runtime visibility via GDB or other debug tools.

PC Tracing Logic in Stage5

- A FIFO (pc_trace_fifo) was instantiated inside stage5.bsv to capture the PC value at every instruction commit.
- Each commit rule (like rl_commit_reg, rl_commit_ld, rl_commit_st, etc.) in the writeback stage was modified to enqueue the exec_pc into this FIFO.
- A new interface method pc_trace_out was added to expose this FIFO to outer modules.

Propagation to Top-Level Modules

- The pc_trace_out interface was propagated upward:
    - From stage5.bsv to riscv.bsv
    - Then to ccore.bsv through the pipe_ifc bundle

- In ccore.bsv, a new rule rl_write_pc_trace was written. It dequeues PC values from pc_trace_out and writes them sequentially into a trace buffer memory.

BRAM Instantiation for PC Storage

- A new Block RAM (BRAM) module named trace_bram was instantiated in soc.bsv.
- This BRAM was memory-mapped to a fixed address range: Base Address: 0xA0000000
- The trace address (trace_addr) is an offset within this region where PCs are stored.
- trace_bram was added as a new AXI slave, and:
    - The Num_Slaves parameter in the AXI interconnect was increased.
    - A new slave number Trace_slave_num was defined in Soc.defines.

Sequential PC Storage

- PCs are written to the BRAM using the AXI Master interface from ccore.bsv in the rl_write_pc_trace rule.

- The address is incremented after every write, ensuring sequential logging of commit PCs.

Benefits of BRAM-based PC Logging

- Since the BRAM is memory-mapped, it can be read directly via:
    - GDB in simulation (through OpenOCD)
    - Verilator logs or direct memory access
- This enables easy tracing of instruction flow for debug or analysis without external interfaces like UART.

3.2 MODULE MODIFICATIONS :

- **stage5.bsv** : Added FIFO logic, enqueue rule for PC, and interface pc_trace_out.
- **riscv.bsv** : connected pc_trace_out interface from stage5 and exposed to ccore.
- **ccore.bsv**: Instantiated an AXI master to write PC data, added logic to incrementally write to BRAM.
- **Soc.bsv**: Created and connected a BRAM module trace_bram to the AXI interconnect.
- **Soc.defines**: Updated constants to define the new slave number and base

In stage5.bsv:

1. pc_trace_fifo.enq(zeroExtend(fuid.pc)) : It enqueues (stores) the current program counter (PC) value from fuid.pc into the FIFO pc_trace_fifo, after zero-extending it to match the FIFO's width.
2. interface pc_trace_out = toGet(pc_trace_fifo) : This exposes the PC trace FIFO as a Get interface so that other modules can read committed PC values from it.

In ccore.bsv:

1. rule rl_write_pc_trace; : This reads a committed PC from the FIFO and writes it to the memory-mapped BRAM at a sequential address for external access.

In Soc.bsv:

1. Ifc_bram_axi4#(`paddr, ...) trace_bram <- mkbram_axi4(...) : It instantiates a memory-mapped BRAM module that can be accessed over the AXI4 bus for storing or reading data like PC trace values.
2. mkConnection(fabric.v_to_slaves[`Trace_slave_num], trace_bram.slave) : It connects the trace BRAM as a memory-mapped slave device to the AXI4 interconnect at the specified slave index.

CHAPTER 4

TESTING AND RESULTS

4.1 GDB VALIDATION

To validate the PC tracing mechanism, a simple RISC-V test program was compiled and loaded into the simulation environment using Verilator. This test program, designed for predictable behavior, was converted into a .mem file using elf2hex and executed in the SoC. During simulation, GDB was connected via OpenOCD, allowing direct inspection of memory-mapped regions. The traced PC values, stored in BRAM and exposed at address 0xA0000000, were accessed using GDB commands such as x/20wx 0xA0000000. These values were then cross-verified against the expected sequence of instruction addresses in the program. The successful match between logged PCs and the committed instruction flow confirmed the correctness of PC extraction at the writeback stage and its storage in BRAM, thereby validating the entire trace path from core to memory to GDB observation.

4.2 ANALYSIS

The system's effectiveness was validated through multiple checks that were closely linked to ensure reliability. It was verified that PC values from the writeback stage were correctly and sequentially stored in BRAM without errors. This reliable capture allowed seamless trace reading via GDB, where the logged PCs were read from the memory-mapped BRAM address space using inspection commands. Furthermore, the integration of the BRAM modules proved efficient, introducing negligible overhead to the pipeline due to their lightweight nature, making this PC tracing mechanism both accurate and non-intrusive.

CHAPTER 5

CONCLUSION

This project successfully implements a PC tracing mechanism within the Shakti C-Class RISC-V core to facilitate runtime instruction tracking for debugging and analysis. By tapping the Program Counter (PC) during the writeback stage of the pipeline and routing it through a FIFO to a dedicated BRAM via the AXI master interface, we enabled real-time logging of committed instruction addresses. The integration of this mechanism required careful modification across several modules, including stage5.bsv, riscv.bsv, ccore.bsv, soc.bsv, and the memory system. With the BRAM memory-mapped to a known address range (0xA0000000), this setup allows GDB and software tools to read back the PC values seamlessly. Testing validated that the PC logging works reliably in simulation using a three-window environment (Verilator, OpenOCD, and GDB). This work lays the foundation for deeper runtime analysis, performance evaluation, and trace-based debugging in custom processor pipelines.

REFERENCES

[1] https://github.com/vyomasystems/uptickpro_examples/tree/main

[2] https://gitlab.com/shaktiproject/cores/c-class

[3] https://gitlab.com/shaktiproject/uncore/devices/-/tree/master

[4] C-Class Core Generator — C-Class Core Generator beta-4.4.0 documentation

[5] c-class-4.6.0-debug-build-steps - Google Docs

[6] Guide for 3 Window Debugging - Google Docs