RISC-V 64-bit Processor Implementation

RV64I + Zba Extension

1 Design Overview

This project implements a **5-stage pipelined** processor supporting the RV64I base instruction set and the **Zba (Address Generation)** extension. The design features a decoupled memory architecture with separate instruction and data interfaces, and a complete data forwarding unit to maintain high throughput.

2 Microarchitecture & Zba Logic

The processor utilizes a classic Fetch-Decode-Execute-Memory-Writeback pipeline:

- **Zba Extension:** Specialized ALU hardware implements `sh1add`, `sh2add`, and `sh3add`. These instructions perform a 1, 2, or 3-bit left shift on a source register before adding it to another, significantly accelerating pointer arithmetic for array indexing.
- **Hazard Management:**
  - **Data:** EX-to-EX and MEM-to-EX forwarding units resolve RAW hazards without stalls.
  - **Control:** A single-cycle pipeline flush (NOP insertion) is triggered for jump instructions to ensure correct execution flow.

3 Implementation Summary

- **riscv_processor.sv**: Core RTL featuring a 32-entry 64-bit register file with synchronous reset and the Zba-enabled ALU.
- **Memory Subsystem**: `instruction_memory.sv` (16KB) and `data_memory.sv` (8KB) provide byte-addressable access for the pipeline.
- **Verification**:
  - **test.c**: A targeted C suite verifying fundamental arithmetic, branching, and Zba-specific operations.
  - **tb_processor.sv**: A self-checking SystemVerilog testbench that validates final architectural states and monitors memory transactions.
- **Build System**: `build_commands.txt` and `Makefile` document the toolchain flags (using `-march=rv64i_zba`) and Verilator simulation flow.

4 Design Refinements

- **Initialization:** Implemented a robust reset sequence for all registers to prevent undefined startup states.
- **Software Co-design:** Leveraged inline assembly in the test program to ensure specific Zba opcodes were executed without being optimized away by the compiler.
- **Efficiency:** Selected a 5-stage pipeline to balance clock frequency with design complexity, meeting all challenge requirements for a clean, synthesizable implementation.

5 Results

The implementation successfully passes the test suite. Verification confirms that the ALU correctly computes Zba-specific address offsets and that the pipeline handles dependencies through forwarding, achieving functional correctness across the RV64I + Zba instruction set.