C-CLASS CORE PERFORMANCE ANALYSIS: L1 I-CACHE AND D-CACHE

SUMMER INTERNSHIP REPORT

Submitted by

RAMYA S -2022105015

MRUTHULA R- 2022105531

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

JUNE-JULY 2025

ANNA UNIVERSITY: CHENNAI 600025

BONAFIDE CERTIFICATE

Certified that this report titled as, is the Bonafide work of **7th Semester** who carried out the project work for **Summer Internship**, in the month of June-July 2025under 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

Dr.M.ABhagyaveni

HEAD OF THE DEPARTMENT

Professor

Department of ECE

College of Engineering Guindy

Anna University

Chennai-600025

SIGNATURE

Nitya Ranganathan

Sriram

IIT Madras Mentors

ACKNOWLEDGEMENT

I express my sincere gratitude to the Dean,**Dr.P.Hariharan**, Professor, College of Engineering, Guindy for his support throughout the project.

I extend my heartfelt appreciation to my Head of the Department, **Dr.M.ABhagyaveni,** Professor, Department of Electronics & Communication Engineering for her enthusiastic support and guidance throughout the project.

I am immensely grateful to my project supervisor and co-ordinator, **Nitya Ranganathan and Sriram** (IIT Madras) for their unwavering assistance, patience, valuable guidance, technical mentorship, and encouragement in our project.

Certificate page

PROBLEM STATEMENT:

C-Class core performance analysis: L1 I-Cache and D-Cache. Vary some configuration knobs in the parameter files and evaluate on risc-v benchmarks, coremark and other microbenchmarks given by Shakti team. Compare performance across configurations.

ABSTRACT:

This project focuses on analyzing the performance of the Shakti C-Class RISC-V core by varying configuration parameters of the Level-1 Instruction Cache (I-Cache) and Data Cache (D-Cache). By modifying cache-related knobs such as associativity, block size, and replacement policies in the parameter YAML files, we evaluate their impact on core performance. The analysis is conducted using standard RISC-V benchmarks, CoreMark, and microbenchmarks provided by the Shakti team. Performance metrics are collected and compared across multiple configurations to identify optimal cache settings for improved efficiency and speed.

PROJECT MOTIVATION AND OVERVIEW:

This project presents a detailed performance analysis of the Shakti C-Class RISC-V core, with a particular focus on the Level-1 Instruction Cache (I-Cache) and Data Cache (D-Cache) subsystems. Efficient cache design is crucial for achieving high performance in processor cores, especially in embedded and real-time systems. The study investigates how key cache parameters—such as associativity, block size, and replacement policies—affect the core's execution efficiency and throughput. These configuration knobs are varied by editing the parameter YAML files that define the C-Class core's cache architecture.

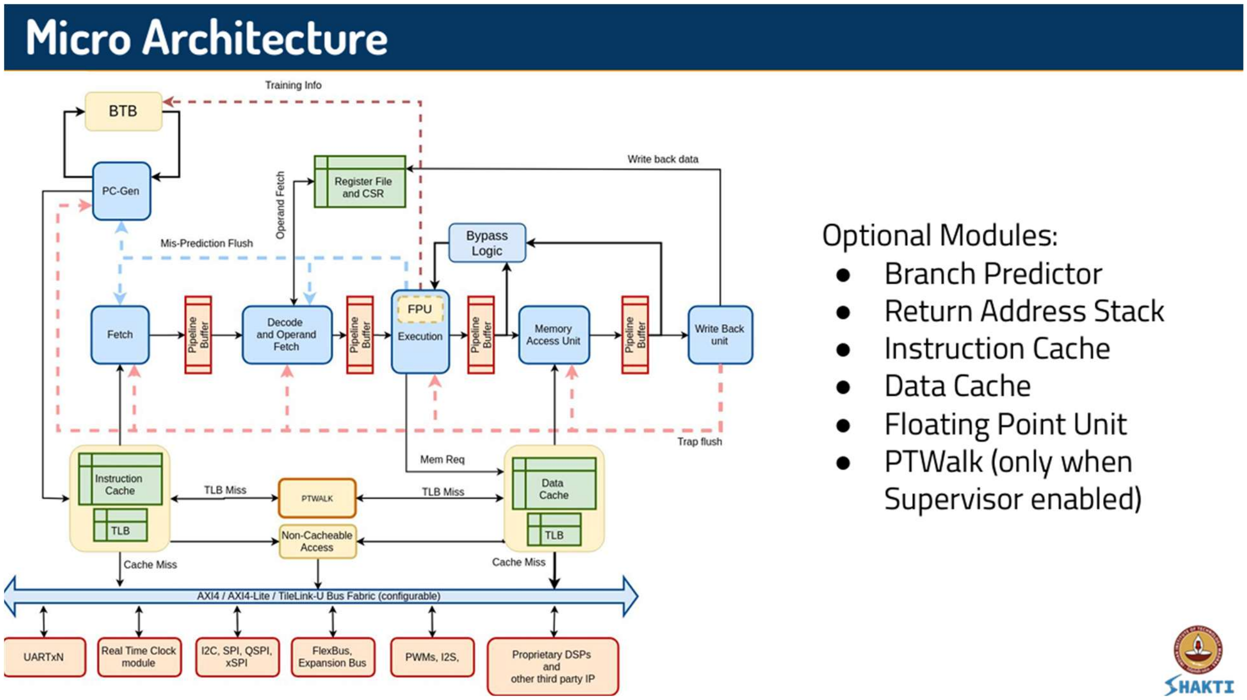
To quantify the impact of these changes, we employ a range of standard RISC-V benchmarks, including the industry-standard CoreMark suite and additional microbenchmarks curated by the Shakti team. These benchmarks are selected to stress different aspects of the processor and memory hierarchy, enabling a well-rounded evaluation of performance under varying workloads. Simulations are run for multiple cache configurations, and key performance metrics.

The results from this study highlight the trade-offs between cache complexity and performance gains, offering insights into which cache configurations yield the best performance for specific application domains. Ultimately, the findings aim to guide future optimization efforts in Shakti core design by identifying cache settings that enhance both efficiency and computational speed without incurring excessive hardware overhead.

SHAKTI C-CLASS CORE:

The Shakti C-Class is a 64-bit in-order RISC-V core developed by the Shakti Processor Project at IIT Madras. It implements the RV64GC ISA, supporting standard base and optional extensions including integer (I), multiplication/division (M), atomic (A), floating-point (F and D), compressed (C), and supervisor mode (S-mode). Designed using Bluespec SystemVerilog (BSV), the C-Class core targets mid-range embedded systems and supports operating systems like Linux.

It features a 6-stage in-order pipeline, supports virtual memory through MMU, and includes configurable Level-1 instruction and data caches. The core is highly modular and parameterizable, enabling architectural experiments such as cache tuning, pipeline changes, and memory hierarchy customization. Its open-source nature and compatibility with standard RISC-V tools make it an ideal platform for research, academic, and industrial development.



I-CACHE AND D-CACHE:

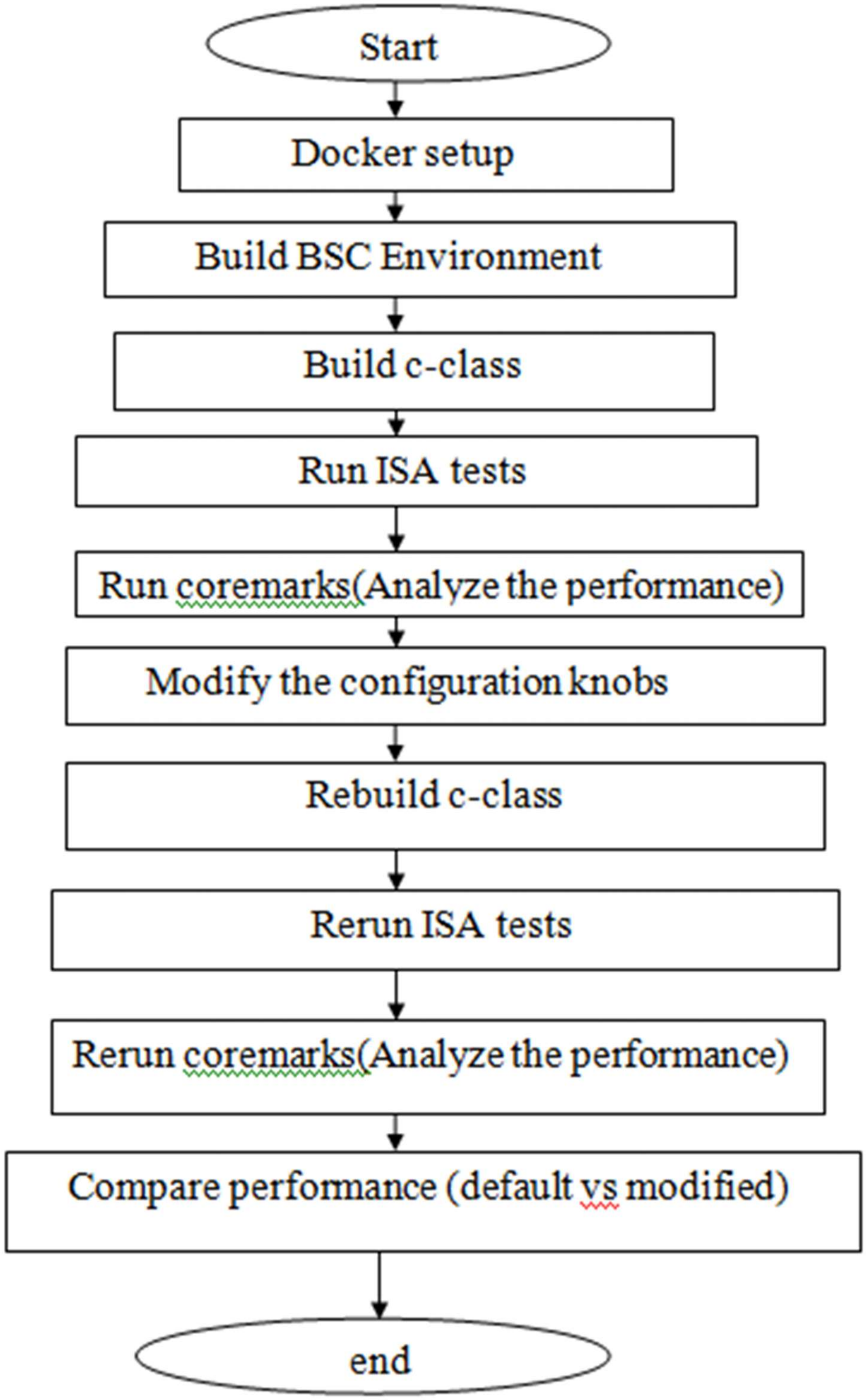
The efficiency of a processor is significantly influenced by its memory hierarchy, particularly the performance of the Level-1 (L1) cache subsystem. In modern embedded systems, L1 caches play a critical role in minimizing memory access latency and maintaining pipeline throughput. The Shakti C-Class, a 64-bit RISC-V core developed as part of the Shakti processor family, adopts a Harvard architecture, incorporating physically separate L1 Instruction (I-Cache) and Data (D-Cache) units. These caches form the first level of data and instruction storage interfaced with the processor pipeline.

The Instruction Cache (I-Cache) is responsible for storing executable instructions fetched from main memory. In the default configuration of the Shakti C-Class core, the I-Cache is implemented as a 4-way set-associative cache comprising 64 sets with a block size of 16 bytes. It employs a RANDOM replacement policy for cache line eviction. This structure is optimized for spatial locality in instruction access patterns and supports seamless integration with the fetch stage of the pipeline.

The Data Cache (D-Cache) is designed to support both read and write operations during load and store transactions. It also features a 4-way set-associative structure with 64 sets, but uses a smaller block size of 8 bytes to better accommodate fine-grained data accesses. A Round-Robin (RR) replacement strategy is employed, ensuring even distribution of evictions across sets. Internal buffering mechanisms, such as load, store, and issue buffers, are also present in the D-Cache to improve memory-level parallelism and reduce data hazards.

These cache configurations are defined in platform-specific YAML parameter files and can be flexibly modified during hardware generation. Key configurable attributes include block size, associativity, number of sets, and replacement policy. The ability to adjust these parameters allows for detailed microarchitectural exploration and performance tuning.

FLOW CHART:



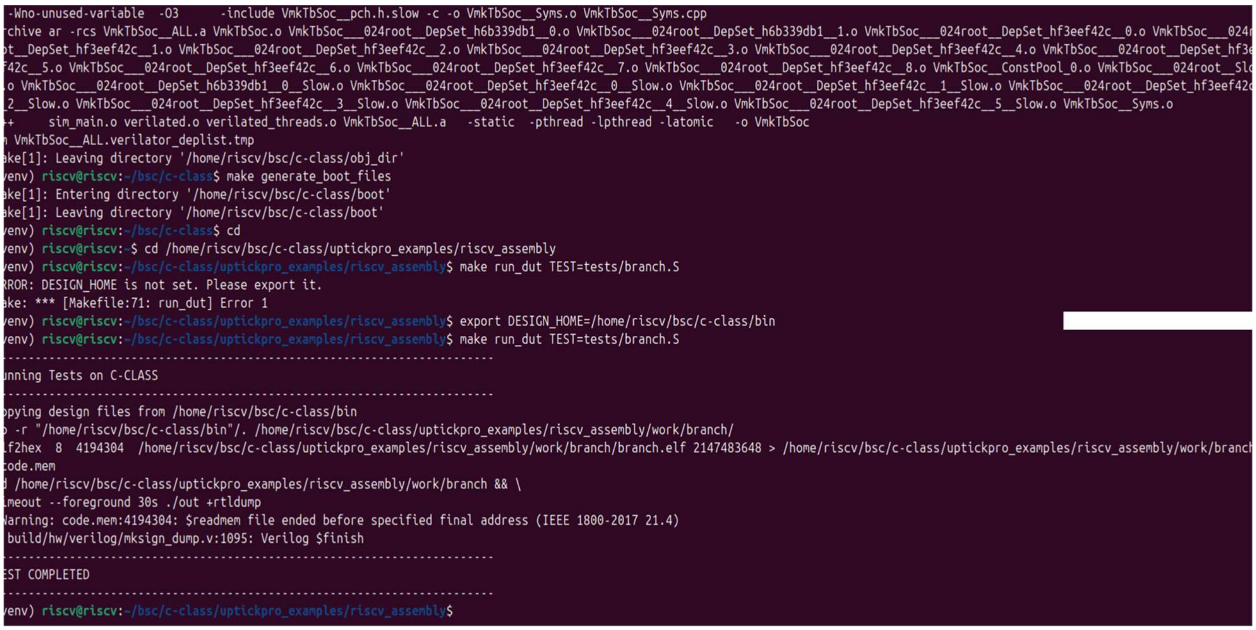
Start
Docker setup
Build BSC Environment
Build c-class
Run ISA tests
Run coremarks(Analyze the performance)
Modify the configuration knobs
Rebuild c-class
Rerun ISA tests
Rerun coremarks(Analyze the performance)
Compare performance (default vs modified)
end

DEFAULT CONFIGURATION:

```yaml
icache_configuration:
  instantiate: true
  sets: 64
  word_size: 4
  block_size: 16
  ways: 4
  replacement: RANDOM
  fb_size: 4
  ecc_enable: false
  one_hot_select: false
dcache_configuration:
  instantiate: true
  sets: 64
  word_size: 8
  block_size: 8
  ways: 4
  fb_size: 9
  sb_size: 2
  lb_size: 4
  ib_size: 2
  replacement: RR
  ecc_enable: false
  one_hot_select: false
  rwports: '1rw'
```

this is the default configuration of the core64.yaml file for that we built a class and ran a basic isa test .

ISA TEST:



```
-Wno-unused-variable  -O3      -include VmkTbSoc__pch.h.slow -c -o VmkTbSoc__Syms.o VmkTbSoc__Syms.cpp
rchive ar -rcs VmkTbSoc__ALL.a VmkTbSoc.o VmkTbSoc___024root__DepSet_h6b339db1__0.o VmkTbSoc___024root__DepSet_h6b339db1__1.o VmkTbSoc___024root__DepSet_hf3eef42c__0.o VmkTbSoc___024r
ot__DepSet_hf3eef42c__1.o VmkTbSoc___024root__DepSet_hf3eef42c__2.o VmkTbSoc___024root__DepSet_hf3eef42c__3.o VmkTbSoc___024root__DepSet_hf3eef42c__4.o VmkTbSoc___024root__DepSet_hf3e
f42c__5.o VmkTbSoc___024root__DepSet_hf3eef42c__6.o VmkTbSoc___024root__DepSet_hf3eef42c__7.o VmkTbSoc___024root__DepSet_hf3eef42c__8.o VmkTbSoc__ConstPool_0.o VmkTbSoc___024root__Slo
.o VmkTbSoc___024root__DepSet_h6b339db1__0__Slow.o VmkTbSoc___024root__DepSet_hf3eef42c__0__Slow.o VmkTbSoc___024root__DepSet_hf3eef42c__1__Slow.o VmkTbSoc___024root__DepSet_hf3eef42c
_2__Slow.o VmkTbSoc___024root__DepSet_hf3eef42c__3__Slow.o VmkTbSoc___024root__DepSet_hf3eef42c__4__Slow.o VmkTbSoc___024root__DepSet_hf3eef42c__5__Slow.o VmkTbSoc__Syms.o
++    sim_main.o verilated.o verilated_threads.o VmkTbSoc__ALL.a   -static  -pthread -lpthread -latomic   -o VmkTbSoc
n VmkTbSoc__ALL.verilator_deplist.tmp
ake[1]: Leaving directory '/home/riscv/bsc/c-class/obj_dir'
venv) riscv@riscv:~/bsc/c-class$ make generate_boot_files
ake[1]: Entering directory '/home/riscv/bsc/c-class/boot'
ake[1]: Leaving directory '/home/riscv/bsc/c-class/boot'
venv) riscv@riscv:~/bsc/c-class$ cd
venv) riscv@riscv:~$ cd /home/riscv/bsc/c-class/uptickpro_examples/riscv_assembly
venv) riscv@riscv:~/bsc/c-class/uptickpro_examples/riscv_assembly$ make run_dut TEST=tests/branch.S
RROR: DESIGN_HOME is not set. Please export it.
ake: *** [Makefile:71: run_dut] Error 1
venv) riscv@riscv:~/bsc/c-class/uptickpro_examples/riscv_assembly$ export DESIGN_HOME=/home/riscv/bsc/c-class/bin
venv) riscv@riscv:~/bsc/c-class/uptickpro_examples/riscv_assembly$ make run_dut TEST=tests/branch.S
---------------------------------------------------------------------
unning Tests on C-CLASS
---------------------------------------------------------------------
opying design files from /home/riscv/bsc/c-class/bin
p -r "/home/riscv/bsc/c-class/bin"/. /home/riscv/bsc/c-class/uptickpro_examples/riscv_assembly/work/branch/
lf2hex  8  4194304  /home/riscv/bsc/c-class/uptickpro_examples/riscv_assembly/work/branch/branch.elf 2147483648 > /home/riscv/bsc/c-class/uptickpro_examples/riscv_assembly/work/branch
code.mem
d /home/riscv/bsc/c-class/uptickpro_examples/riscv_assembly/work/branch && \
imeout --foreground 30s ./out +rtldump
Warning: code.mem:4194304: $readmem file ended before specified final address (IEEE 1800-2017 21.4)
 build/hw/verilog/mksign_dump.v:1095: Verilog $finish
---------------------------------------------------------------------
EST COMPLETED
---------------------------------------------------------------------
venv) riscv@riscv:~/bsc/c-class/uptickpro_examples/riscv_assembly$
```

this is the pics show that we successfully ran isa test

APPLOG FILE:

```
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 14929864
Total time (secs): 14.929863
Iterations/Sec   : 2.679194
Iterations       : 40
Compiler version : riscv64-unknown-elf-15.1.0
Compiler flags   : -mcmodel=medany -DCUSTOM -DPERFORMANCE_RUN=1 -DMAIN_HAS_NOARGC=1 -DHAS_STDIO -DHAS_PRINTF -DHAS_TIME_H -DUSE_CLOCK -DHAS_FLOAT -DITERATIONS=40 -O3 -fno-unsafe-math-optimizations -fno-tree-loop-vectorize -fno-strict-aliasing -fgnu89-inline -fno-common -funroll-loops -finline-functions -fselective-scheduling -falign-functions=4 -falign-jumps=4 -falign-loops=4 -finline-limit=1000 -nostartfiles -nostdlib -ffast-math -fno-builtin-printf -mabi=lp64d -march=rv64imafdc -mexplicit-relocs --param max-unroll-times=2
Memory location  : STACK
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0x65c5
Correct operation validated. See README.md for run and reporting rules.
CoreMark 1.0 : 2.679194 / riscv64-unknown-elf-15.1.0 -mcmodel=medany -DCUSTOM -DPERFORMANCE_RUN=1 -DMAIN_HAS_NOARGC=1 -DHAS_STDIO -DHAS_PRINTF -DHAS_TIME_H -DUSE_CLOCK -DHAS_FLOAT -DITERATIONS=40 -O3 -fno-unsafe-math-optimizations -fno-tree-loop-vectorize -fno-strict-aliasing -fgnu89-inline -fno-common -funroll-loops -finline-functions -fselective-scheduling -falign-functions=4 -falign-jumps=4 -falign-loops=4 -finline-limit=1000 -nostartfiles -nostdlib -ffast-math -fno-builtin-printf -mabi=lp64d -march=rv64imafdc -mexplicit-relocs --param max-unroll-times=2 / STACK
```

RTLDUMP FILE:

```
core   0: 3 0x000000008000453e (0x3003031b) x6 0x0000000000011300
core   0: 3 0x0000000080004542 (0x00c31503) x10 0x0000000000000001 mem 0x000000000001130c
core   0: 3 0x0000000080004546 (0x8905) x10 0x0000000000000001
core   0: 3 0x0000000080004548 (0xd975)
core   0: 3 0x000000008000454a (0xc0202573) x10 0x0000000000c6991e
core   0: 3 0x000000008000454e (0xc0002573) x10 0x0000000000f712cc
core   0: 3 0x0000000080004552 (0xb0302573) x10 0x000000000003658f
core   0: 3 0x0000000080004556 (0xb0402573) x10 0x0000000000024f07
core   0: 3 0x000000008000455a (0xb0502573) x10 0x000000000020c392
core   0: 3 0x000000008000455e (0xb0602573) x10 0x000000000005bcd8
core   0: 3 0x0000000080004562 (0x00020537) x10 0x0000000000020000
```

PERFORMANCE EVALUATION:

The RTL simulation output provides several hardware performance counters that quantify the execution characteristics of the program running on the Shakti C-Class core. The following values were extracted and analyzed for performance insights:

Instructions Retired (instret): 0x0000000000c6991e

Total Clock Cycles (cycle): 0x0000000000f712cc

Branch Mispredictions (hpmcounter3): 0x000000000003658f

Jump Instructions Executed (hpmcounter4): 0x0000000000024f07

Number of Branches (hpmcounter5): 0x000000000020c392

Multiply/Divide Operations (hpmcounter6): 0x000000000005bcd8

MODIFIED YAML FILE:

Open

core64.yaml × core_mai


```yaml
ISA: RV64IM
s_extension:
  itlb_size: 4
  dtlb_size: 4
total_events : 30
#dtvec_base: 256
iepoch_size: 2
m_extension:
  mul_stages : 1
  div_stages : 32
icache_configuration:
  instantiate: true
  on_reset: enable
  sets: 64
  word_size: 4
  block_size: 16
  ways: 1
  replacement: RR
  fb_size: 4
  ecc_enable: false
  one_hot_select: false
dcache_configuration:
  instantiate: true
  on_reset: enable
  sets: 64
  word_size: 8
  block_size: 8
  ways: 1
  fb_size: 8
  sb_size: 2
  replacement: LRU
  ecc_enable: false
  one_hot_select: false
  rwports: 1
reset_pc: 4096
physical_addr_size: 32
bus_protocol: AXI4
debugger_support: false
no_of_triggers: 0
bsc_compile_options:
  test_memory_size: 33554432
  assertions: true
  trace_dump: true
  compile_target: 'sim'
  suppress_warnings: ["none"]
  verilog_dir: build/hw/verilog
  build_dir: build/hw/intermediate
```

RESULT:

We successfully built the Shakti C-Class core and validated its functionality by executing the RISC-V ISA test suite. Initial performance evaluation was conducted using CoreMark, based on the default configuration provided in the core64.yaml file.

To analyze the impact of cache parameters on performance, we modified the core64.yaml configuration — specifically adjusting the number of sets, associativity (ways), and replacement policy for both L1 instruction and data caches. After rebuilding the core with the updated configuration, we re-ran the ISA tests and benchmarks, all of which executed without errors.

However, the performance outputs observed in rtl1.dump and out remained the same as those obtained with the default configuration, indicating that the performance did not change despite the cache modifications.

REFERENCES AND LINKS

- Shakti C-Class Core (GitLab Repository)
  https://gitlab.com/shaktiproject/cores/c-class

- C-Class Sample Configuration File
  https://gitlab.com/shaktiproject/cores/c-class/-/blob/master/sample_config/c64/core64.yaml?ref_type=heads

- Shakti Benchmark Repository – CoreMark Branch
  https://gitlab.com/shaktiproject/cores/benchmarks/-/tree/cclass-counter-prints?ref_type=heads

- github repo https://github.com/ramyamruthula/RISCV.git