PHILIPPINE ENGINEERING JOURNAL
PEJ 2026; Vol. 47, No. 1:19-33

Received: 3 December 2025
Received in Revised Form: 04 April 2026
Accepted: 10 May 2026

Implementation and Performance Analysis of a Dual-core RV32IMC Processor with Split L1 and Shared L2 Data Caches

Daniel Miguel R. Hora, Don Jeric L. Monderin, Jan Jerald C. Reyes, Allen Jason A. Tan, Anastacia B. Alvarez, Adelson N. Chua*
Electrical and Electronics Engineering Institute,
University of the Philippines Diliman, Quezon City, Metro Manila, Philippines
**Corresponding author: adelson.chua@eee.upd.edu.ph*

Abstract *– There is a growing trend toward performing data analytics closer to embedded devices, in order to compress high-bandwidth raw data into more meaningful and compact information. Multicore and many-core architectures have emerged as scalable hardware platforms that meet increasing computational demands while maintaining energy efficiency. However, their performance is often limited by memory access bottlenecks, as simple multicore designs compete for shared memory resources. To address this, cache memory is typically introduced to reduce memory traffic by servicing a significant portion of memory requests locally. This work presents a proof-of-concept dual-core RISC-V processor system, implemented on the Nexys Video FPGA board featuring a Xilinx Artix-7 (XC7A200T) device. The design features private level-one data caches per core and a shared level-two data cache, with cache coherence maintained via a snooping-based protocol. To evaluate performance scalability, six parallelizable algorithms were benchmarked. The system operates at 30 MHz and consumes an average of 1.001 W of power, with the dual-core processor system only consuming 46 mW of power. Despite operating at a reduced frequency of 30 MHz compared to the 50 MHz single-core baseline, the dual-core system achieved an average speedup of 1.64× across the evaluated programs.*

Keywords*: Multicore, processor, RISC-V, Cache memory*

I. INTRODUCTION

The increasing demand for processing large volumes of data in real-time has made multicore processor architectures a fundamental component in many computing systems. [1]. For example, modern edge computing devices, like surveillance cameras or industrial monitoring units, now require multiple image or camera sensors to observe a scene and provide more complete data for a more accurate representation of the environment. Each node is typically responsible for collecting, processing, and transmitting data to a central unit for further analysis.

As applications scale to support higher data throughput, the computational load increases. Traditional single-core systems become insufficient for meeting real-time processing requirements, especially when systems are expected to filter, aggregate, or analyze data locally before transmission [1]. To address this limitation, multicore processors are introduced within

sensor nodes to enable parallel execution of tasks such as signal processing, compression, or event detection.

While multicore processors offer significant performance gains, they also introduce challenges that make designing efficient systems more complex. Key considerations include process synchronization, data coherence, and task scheduling, all of which must be carefully managed [2]. These challenges span multiple levels, from hardware design to software architecture, requiring a coordinated approach to ensure correct and efficient system behavior. To alleviate bottlenecks in memory resources from contending cores and reduce access latency, a memory hierarchy is introduced, commonly featuring cache systems [3]. Caches act as fast, intermediate storage close to the processor cores, holding recently or frequently accessed data to reduce the need for repeated access to slower main memory.

In a multicore system, where each core may independently access and modify data, maintaining data consistency becomes a fundamental requirement. This is handled through cache coherence protocols, which ensure that all cores observe a consistent view of shared memory. Without these mechanisms, one core may read stale data that has already been modified by another, resulting in erroneous program behavior [2]. Cache coherence protocols must operate in conjunction with the cache hierarchy to coordinate data movement between caches and memory.

Designing multicore systems with an efficient cache hierarchy and coherence mechanism is therefore essential in achieving reliability and speed. Faster execution often means the CPU can spend more time in low-power sleep states. Multicore systems also enable individual cores to operate concurrently with reduced memory latency, while preserving correct and predictable system behavior.

This work builds upon these foundational principles by exploring the design and implementation of a 32-bit homogenous dual-core RISC-V processor system with split L1 data caches and a shared L2 data cache on an FPGA platform. The processor is designed to execute RISC-V programs and is implemented in Verilog, targeting the Artix-7 A200T FPGA—chosen for its high logic capacity and ample external memory support. The work primarily emphasizes execution speed, power efficiency, and resource utilization. The main objective is to demonstrate the performance gains achievable through a dual-core architecture with cache hierarchies, particularly in comparison to a baseline single-core design.

II. REVIEW OF RELATED WORK

2.1. Applications of Multi-core Systems

There are numerous multi-core architectures employing RISC-V cores, each tailored with unique configurations and extensions for their target applications [4]. The hybrid hardware-software solution of Jang et al. [2] aims at simplifying the development of multicore architectures from open-source single-core implementations. Single cores are not inherently designed for multi-core environments, particularly when dealing with shared resources. As such, adapting them typically involves significant redesign efforts, especially concerning cache

coherence. Their work [2] introduces the Temporary Caching (TC) technique, a primarily software-based scheme. The key idea is to enable programmers to temporarily cache data whenever possible, enhancing performance by dynamically caching data that would otherwise require main memory access. Nevertheless, limitations such as the cache-line problem are also discussed in the work. The work addresses this issue with the Temporary Caching Unit (TCU), a dedicated hardware component that manages virtual address mapping and byte-level control to resolve the cache-line problem. Ultimately, the study suggests that complete cache coherence can only be achieved with dedicated hardware solutions. Designers must weigh the trade-offs between implementing dedicated cache-coherency mechanisms and leveraging the TCU's software-based approach, which introduces additional overhead.

The work in [5] introduces a many-core architecture comprising clusters of RISC-V processing elements (PEs) interconnected via a Network-on-Chip (NoC). The architecture emphasizes scalability through a modular and adaptable NoC topology. Another work using a cluster architecture is Mr. Wolf [6]. Mr. Wolf is an SoC that features a cluster of 8 DSP-optimized RISC-V cores supporting RV32IM plus extensions for efficient DSP. The cluster is served by a multi-banked L1 memory. The L1 memory can serve all memory requests in parallel with a single cycle latency and a low average contention rate.

The power efficiency gains achieved in the work of Dogan et al. [7] are acquired from transitioning from a single-core to a multi-core processor architecture for a wireless body sensor network (WBSN) used in personal health systems, such as electrocardiogram (ECG) sensors. The single-core design operated at 1.2 V with a power consumption of 10.4 mW, whereas the multi-core design functioned at 0.7 V, reducing power consumption to just 3.5 mW.

Although ECG signal processing involves relatively low computational complexity suitable for typical low-power embedded microcontrollers, minimizing power consumption remains a critical challenge. Personal health systems, such as ECG sensors, are required to function on a single battery for extended durations. While power optimization techniques can be applied to single-core systems, they often result in degraded performance. A multi-core architecture addresses this trade-off by leveraging parallel computing.

2.2. Cache Systems

In the study conducted by Alidio et al. [8], the researchers sought to enhance the performance of the ARM7 microprocessor—originally equipped with a single-level split cache—by implementing a two-level cache architecture, incorporating a shared L2 cache. The main goal was to improve overall system efficiency by reducing average execution time, primarily through optimal sizing of the L2 cache to limit reliance on main memory.

To evaluate their design, the team benchmarked the system using sorting algorithms such as bubble sort and insertion sort. Results indicated that in configurations relying solely on an L1 cache, frequent cache misses can lead to substantial performance penalties due to costly main memory accesses [8]. Introducing an L2 cache between the L1 cache and main memory addresses this bottleneck by serving as an intermediate buffer. This addition helps lower cache

miss rates and bridges the latency disparity between the fast, limited-capacity L1 cache and the slower main memory.

III. DUAL-CORE RISCV PROCESSOR DESIGN

Figure 1 shows the overall top-level diagram of the work. The diagram shows the main dual-core architecture module (highlighted in yellow) connected to the necessary and optional peripherals for proper integration into an FPGA.

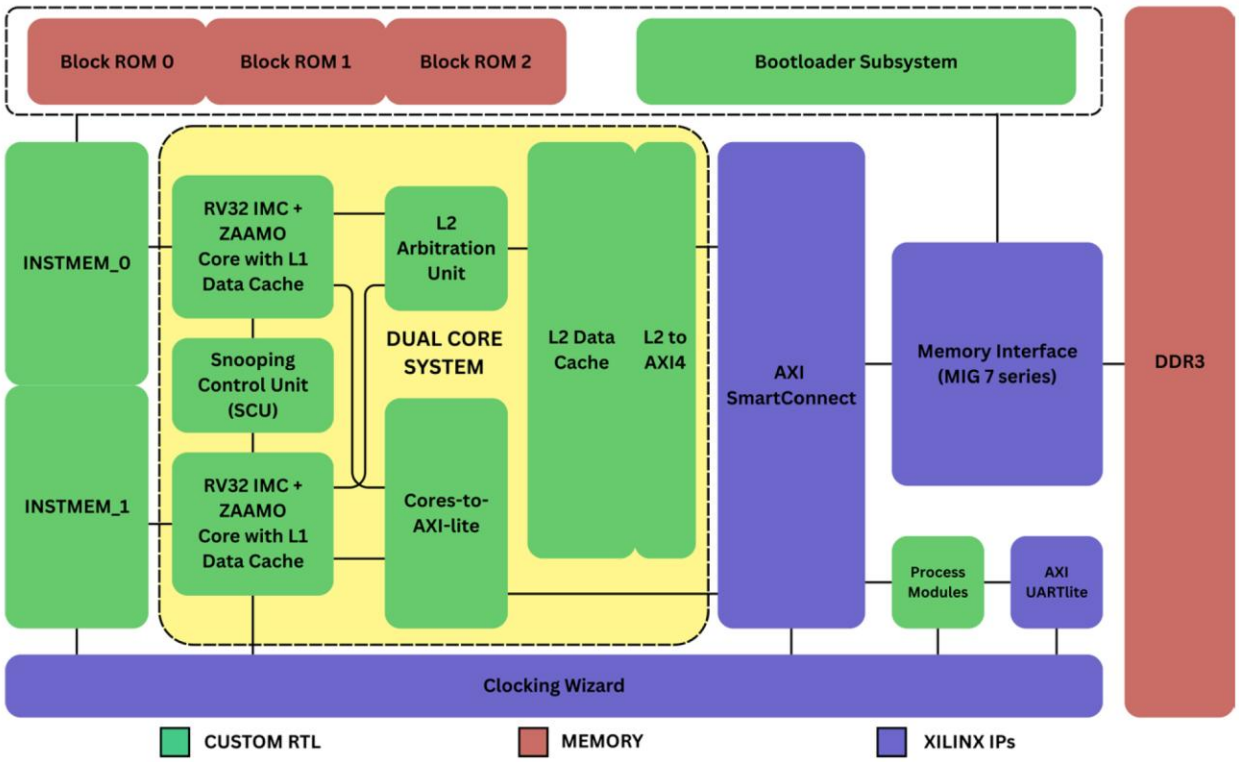

Figure 1. Top-level diagram of the Dual Core Processor FPGA-based System.

This work focuses on implementing a dual-core architecture based on an existing RISC-V core design [9]. The original core is extended to support private L1 data caches and atomic extension, enabling coherent and efficient multiprocessor operation. The processor system architecture includes two RISC-V cores, each with its own cache, a shared L2 data cache, arbitration modules to manage access to shared resources, and AXI4-compatible adapter modules to interface with AXI-based memory and peripheral subsystems.

3.1. RISC-V Core Modifications

The original core was designed to operate at 50 MHz [9]. However, the current design operates at 30 MHz per core due to system-wide timing constraints. The observed drop in maximum operating frequency is primarily attributed to the integration of the L1 cache. To achieve low memory access latency, the cache is designed to operate on the negative edge of the core's clock, effectively offsetting the cycle during which data is fetched. This approach

forces the cache to function within only half of the available clock period's setup and hold times, making timing closure more challenging. The current cache implementation does not optimally meet these tight timing constraints. Nevertheless, this design trade-off was chosen to minimize memory access latency without significantly increasing architectural complexity. Despite the reduced clock frequency, the dual-core system compensates through improved throughput enabled by parallel task execution.

The RISC-V core is modified to support a subset of the Atomic (A) extension [4], specifically the ZAAMO subset. This enables the core to perform atomic memory operations such as atomic OR, AND, XOR, ADD, and SWAP. These operations follow a load-operate-store sequence that guarantees atomicity, meaning the value is loaded from memory, the operation is performed, and the result is written back to memory without interruption by other cores. Such behavior is critical for maintaining correct synchronization between cores and preventing race conditions during concurrent memory accesses. The full implementation of the Atomic (A) extension was not included in order to maintain a smaller core footprint. This subset already provides essential atomic instructions for synchronization while minimizing additional hardware complexity.

The RISC-V cores operate with a region-based memory map, enabling different access paths based on the target address. Memory accesses falling within a parameterizable range, referred to as the cacheable region, are routed through the L1 cache, then to the shared L2 cache, and finally to external memory. Addresses outside this range fall into the non-cacheable region, which is reserved for synchronization flags, critical shared data, and memory-mapped peripherals. Accesses to this region bypass the caching hierarchy entirely and are routed directly to the Cores-to-AXI-Lite module, ensuring a consistent and up-to-date view of memory for all cores.

3.2. Caches

The L1 and L2 caches provide a local, fast memory that is logically and physically closer to the RISC-V cores. The subsystem reduces the average memory access time and the core access contention rate to shared memory.

Both L1 and L2 caches were designed to share the following characteristics:

- They are set-associative caches with a configurable number of ways per set (2-way, 4-way, or 8-way).
- Each block contains a fixed size of four 32-bit words.
- They have a configurable number of sets, determined by the total cache size, block size, and associativity.
- They automatically perform data block refills on cache misses.
- They have a Pseudo Least Recently Used eviction policy.
- They have a Write-Allocate policy in case of a cache miss.

The L1 and L2 caches implement different write policies tailored to their roles in the memory hierarchy. The L1 cache uses a write-through policy, where all write operations are immediately propagated to the next level, which is the L2 cache. This approach simplifies

cache coherence management in a dual-core system because the L1 and L2 caches are tightly coupled; the latency overhead of write-through operations is minimal compared to writing directly to external memory.

In contrast, the L2 cache adopts a write-back policy to reduce the frequency of accesses to the high-latency external DDR3 memory. This strategy improves performance by buffering writes and only committing them to main memory when necessary.

3.3. Snooping Control Unit

To maintain a consistent and coherent memory view across the dual-core RISC-V system, a Snooping Control Unit (SCU) is implemented to enforce the MESI (Modified, Exclusive, Shared, Invalid) cache coherence protocol [10]. The MESI protocol, as implemented in this system, ensures that each cache line can be in one of the following four states:

- Modified: The block is only present in the current cache and contains data not yet written back to memory (dirty).
- Exclusive: The block is present only in this cache and is clean (matches main memory).
- Shared: The block may be held in multiple caches and is clean; any modifications require coordination.
- Invalid: The block is no longer valid and must be reloaded before use.

Upon receiving the snoop result (either a snoop hit or a snoop miss), the SCU issues appropriate upgrade or downgrade commands to adjust the MESI state of the relevant cache blocks. For instance:

- If Core 0 issues a read and the line is not present in Core 1's cache, the SCU instructs Core 0's cache to mark the line as Exclusive.
- If the line is present, both caches transition the block to a Shared state to reflect concurrent access.
- In the case of a write request, the SCU sends an invalidate message to the peer cache, ensuring that only the writing core retains the line in a Modified state, and stale copies in other caches are discarded (Invalid state).

The SCU operates externally to the two RISC-V cores but is tightly coupled to their private L1 caches and the shared L2 cache. This architectural separation allows for concurrent development of the L1 cache logic and coherence mechanisms, while their tight coupling ensures low-latency coordination during memory access events.

The SCU continuously monitors memory access requests from both cores by snooping on a global coherence bus. It listens to address and control signals issued by the cores, parses the accessed memory address into its tag, index, and offset components, and initiates lookups in the peer core's Tag Array. This allows it to determine whether a memory block is already cached by the other core and what MESI state it is currently in.

To prevent inconsistency during overlapping transactions, any coherence state transitions triggered by the SCU are deferred by the cache if it is currently servicing a request. These transitions are applied only when the cache reaches an idle state, maintaining correctness without stalling critical operations.

3.4. Other Peripherals

The design uses custom AXI4 and AXI-Lite adapters to facilitate transactions between the essential components. The adapters allow the dual-core processor to connect to AXI-compatible slave modules.

At the top level, the system architecture integrates several critical components, primarily instantiated from Xilinx Intellectual Property (IP) cores to guarantee robust and reliable operation. As depicted in darker blue hues in Figure 1, these Xilinx IP blocks fulfill key infrastructural roles such as routing, interfacing with the off-chip DDR3, and generating the required clock domains.

IV. RESULTS AND DISCUSSION

4.1. Cache Analysis

The performance of the cache architecture was evaluated using well-known sorting programs, including bubble sort and quick sort, as well as digital signal processing kernels such as finite impulse response (FIR) filters and Fast Fourier Transform (FFT). Based on the execution time analysis illustrated in Figure 2, there is a clear trend indicating that increasing the size and associativity of the L1 cache significantly improves execution performance, particularly when moving from very small configurations to larger sizes. Beyond a certain point, however, the performance gains begin to decrease, with execution times stabilizing for larger L1 configurations.

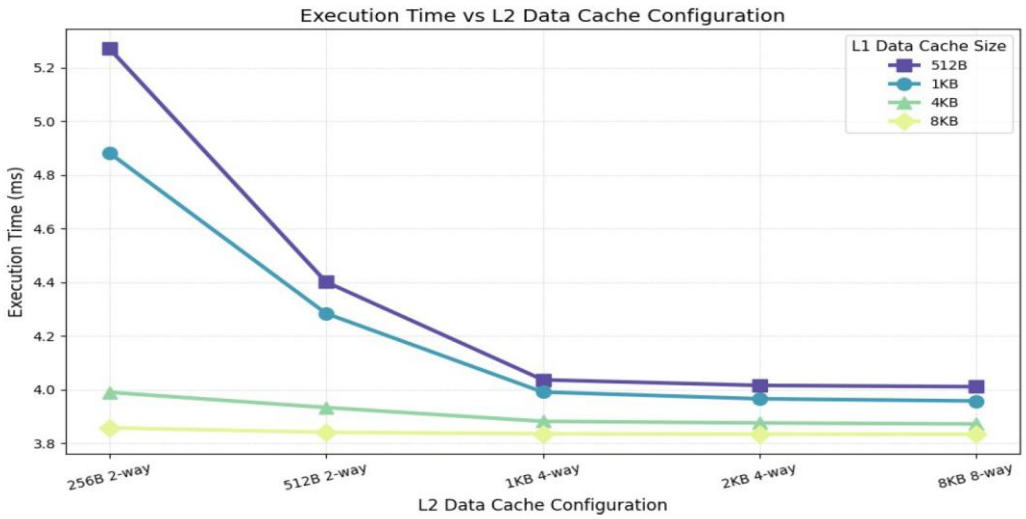

Figure 2. Performance of L1 and L2 cache for Different Configurations and Sizes

Figure 3 illustrates the relationship between L1 cache size and hit rates across both L1 and L2 caches for a dual-core system under varying L2 configurations. As the L1 size increases from 256 bytes to 2048 bytes, the hit rates for both Core 0 and Core 1 steadily improve, with Core 1 maintaining a slight advantage. This growth in L1 hit rate is expected, as larger caches are better equipped to store frequently accessed data, thus reducing memory access latency and improving overall performance. However, this improvement in L1 performance has an inverse effect on L2 hit rates: as L1 becomes more effective at capturing memory accesses, fewer requests reach the L2 cache, leading to a drop in its hit rate. This is because cache hits are mainly absorbed by the L1, with the L2 incurring compulsory misses on the first cache fill. This trend is evident in all L2 configurations, where the hit rates progressively decline with larger L1 sizes.

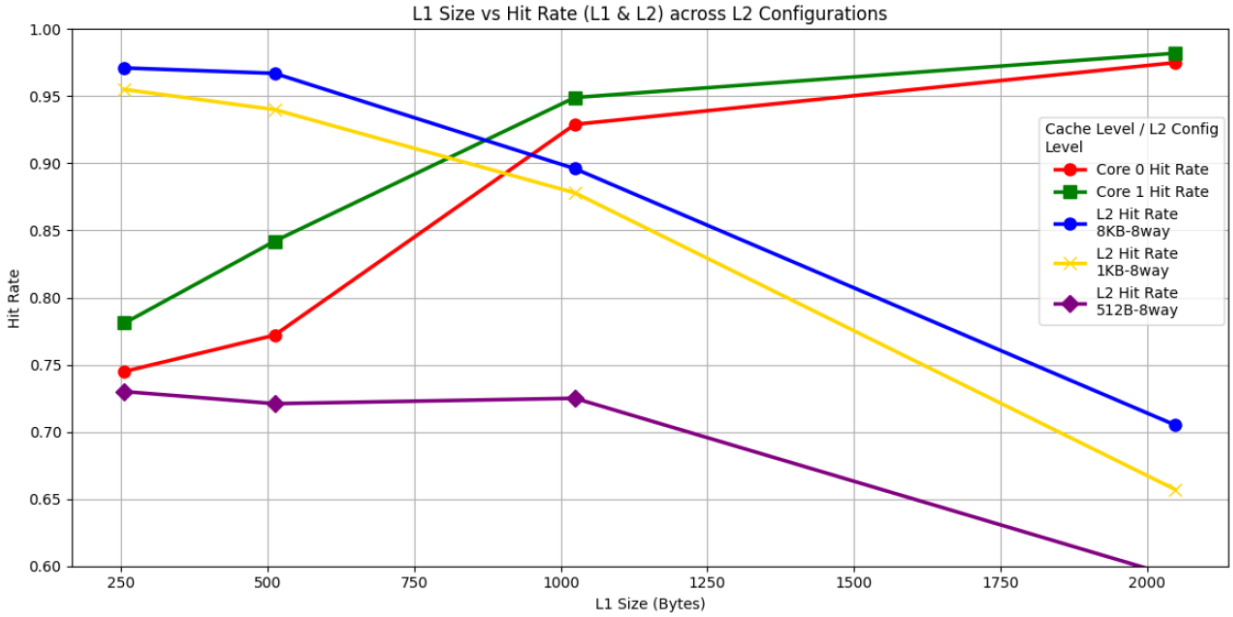

Figure 3. Hit rates of L1 and L2 caches

This behavior highlights a key design trade-off in multi-level caching: while increasing the L1 cache size improves its hit rate and overall system speed, it can also reduce the effectiveness of the L2 cache, which becomes less frequently accessed and thus underutilized. To address this, it is essential to find a "sweet spot" where both L1 and L2 caches achieve high hit rates without excessive hardware cost. Considering the trade-off between performance gains and resource efficiency, this work suggests adopting an L1 cache size of 1KB with 4-way associativity. This configuration offers a strong balance between performance and area efficiency by providing high L1 hit rates while still allowing the L2 cache to contribute meaningfully. For the L2 cache, an 8KB configuration with higher associativity is recommended, as it supports the L1 cache effectively, absorbing accesses that miss L1 while maintaining reasonable area and power overhead. In this configuration, L2 continues to offer performance benefits by caching less frequent but still valuable memory accesses. This pairing (1KB L1 and 8KB L2) demonstrates an optimized hierarchy where both levels of cache are

used efficiently, ultimately improving system performance without excessive hardware complexity.

While increasing cache sizes can improve performance by increasing cache hits, this benefit diminishes beyond a certain point. Excessively enlarging the L1 cache yields only marginal improvements while consuming more area and increasing power consumption. More critically, the cache subsystem remains bound by the physical constraints of the FPGA. As cache size grows, so do fan-out, wire lengths, and net delays—particularly when checking for cache hits. These factors introduce access latency that can negate the primary purpose of caches: fast access to frequently used data. Even with an infinitely large FPGA, arbitrarily increasing cache size has diminishing returns, as the overhead in hardware size and latency may eventually outweigh the gains in performance. Similarly, while a larger L2 cache can moderately reduce overall miss rates, the performance gap it introduces becomes narrower with size, reinforcing the need to balance capacity against timing and efficiency constraints.

4.2. Power Analysis

The configuration of the main architecture used for the power analysis is as follows: a dual-core system, where each core is equipped with a 4-way set-associative 1 KB L1 cache, and both cores share an 8-way set-associative 8 KB L2 cache. The system operates at a clock frequency of 30 MHz. Table 1 shows the relative impact of the different L1 cache sizes on performance (measured through the achievable operating clock frequency) and power consumption. The L1 cache has a more prominent impact on both metrics since it is the memory that the cores directly communicate with. A larger cache size increases the access times, which can result in degraded performance since the system has to compensate through lower clock frequencies to meet the timing constraints. Since both L1 and L2 caches are implemented as registers together with the cores, the impact of L2 cache size on power consumption follows that of the L1 trend. The power consumption of the cores, as will be shown in Figure 4, also tends to be small relative to the overall FPGA power, since the latter is dominated by the IP cores.

Table 1. Impact of Different Cache Sizes on the System Performance and Power Consumption

L1 cache size	Achievable operating clock frequency (MHz)	Power consumption contribution relative to the cores (%)
256B	50 MHz	3.6%
512B	38 MHz	7.5%
1KB	30 MHz	16.0%
2KB	25 MHz	33.5%
4KB	15 MHz	71.1%
8KB	10 MHz	151.3%

For initial program and data storage, the design utilizes three 32-bit wide block ROMs, each with a depth of 4096 words, providing 48 KB of block ROM, which stores the instructions

for both cores as well as the initial data required during system boot. These block ROMs are pre-initialized via the FPGA bitstream, allowing the contents to be loaded directly during configuration without requiring a separate memory loading phase.

The power analysis results presented in this report are obtained using Vivado's Report Power tool under the Implemented Design view. The tool analyzes the post-implementation netlist and estimates power consumption based on inferred switching activities of the internal nets and modules. The tool uses its default assumptions to estimate the power usage across the design.

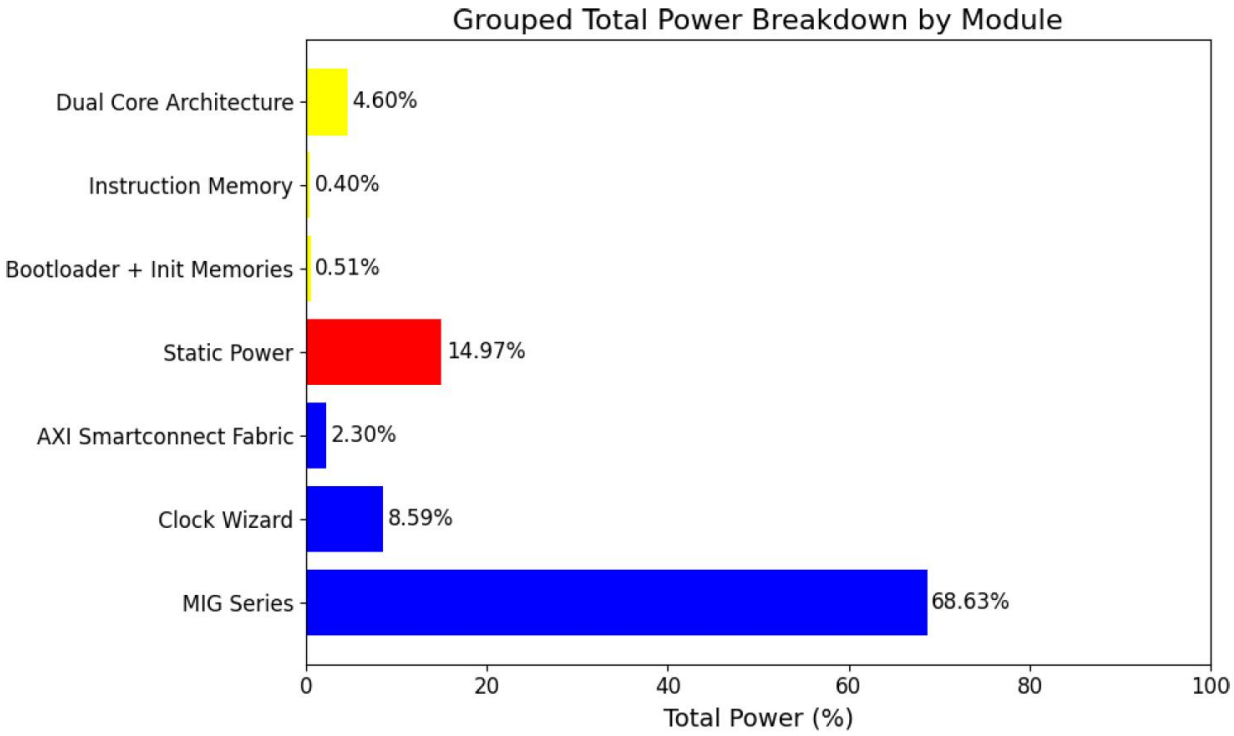

Figure 4. Total Power Breakdown by Module

Figure 4 presents the power breakdown from a module-level perspective. Based on Vivado's Report Power tool, the estimated on-chip power consumption totals 1.001 W. Static power accounts for 0.150 W, representing 14.97% of the total. This portion reflects the leakage and quiescent power required to keep the FPGA's logic resources powered, regardless of activity. Dynamic power is estimated at 0.851 W, comprising 85.03% of the total power.

The graph reveals that the majority of the power is consumed by the generated memory interface module. In contrast, the designed and implemented dual-core subsystem of this work—which includes the two RISC-V cores along with their L1 and L2 caches—accounts for only 4.60% of the total power under typical workloads, or around 46 mW.

4.3. Resource Utilization

Table 1 provides a comparative analysis of resource usage across different processor core configurations. This work builds upon the base RV32IMC core by introducing support for some atomic operations and allowing support for an L1 cache. Despite the inclusion of an additional core and substantial caches, the total resource usage only increased to 1.90× of the original single-core design. This is due to architectural simplifications done on the base core, such as simplifying the branch prediction unit, which originally introduced a rather significant architectural complexity [9].

Table 2. FPGA Resource Utilization Across RISC-V Configurations

Resource	RV32IMC [9]	Dual Core	Dual Core + Caches
LUT	5548	5038	10592
LUTRAM	28	0	15
FF	5670	3482	6392

Table 3. FPGA Resource Utilization Percentage for the Isolated Dual-Core with L1 and L2 Caches in XV7A200T Chip

Resource	Utilization	Available	Utilization (%)
Slice LUTs	10,441	134,600	7.76
Slice Registers	6,392	296,200	2.16
Block RAM Tile	36	365	9.86
DSP	16	740	2.16

As shown in Table 3, the dual-core processor subsystem consumes a relatively small portion of the XC7A200T FPGA fabric, with only 7.76% of the LUTs, 2.16% of the flip-flops, and 9.86% of the BRAM tiles used. The modest footprint highlights the efficiency of the dual-core design, leaving substantial headroom for additional cores, larger caches, or integration of custom accelerators and peripherals.

4.4. Performance

The performance of the dual-core processor is evaluated by measuring the total time required to execute a target algorithm. These measurements are obtained through simulation rather than on physical hardware, as precise real-time measurements are often impractical and less deterministic.

The single-core implementation without caches was able to operate at a maximum operating frequency of 50 MHz, and performance measurements were taken accordingly. In contrast, the dual-core configurations, with and without caches, were restricted to a maximum frequency of 30 MHz, in accordance with the design specifications and timing closure limitations.

To facilitate controlled and observable testing within the Vivado simulation environment, the DDR3 memory and its interface are replaced with a simplified memory model instantiated

directly in the testbench. This substitution enables easier monitoring of memory transactions and content verification, without the overhead of simulating the full DDR3 protocol. Requests still propagate through the full hierarchy, including L1 and L2 caches and the AXI interconnect, before reaching this endpoint.

All other architectural components remain consistent with the implemented design: two RISC-V cores, each with a private 4-way 1 KB L1 cache, a shared 8-way 8 KB L2 cache, and AXI SmartConnect interconnect. This preserves realistic cache behavior, contention, and routing delays across the memory hierarchy. To test cache-less performance, the cores force all memory access to be routed to the non-cacheable region.

Figure 5 presents the performance results of the dual-core processor across various benchmark programs.

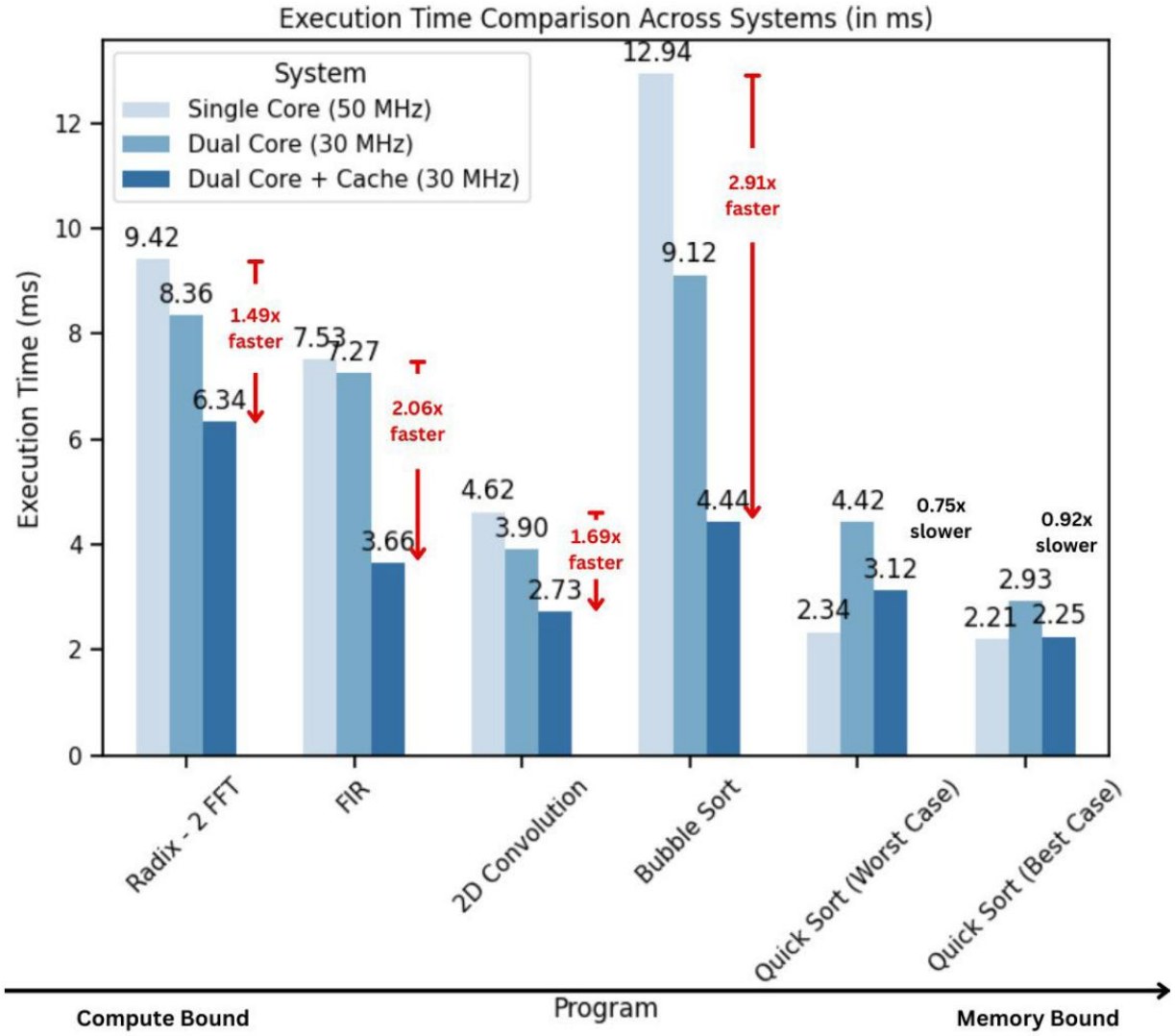

Figure 5. Performance Results across Different Programs

Six parallelizable common algorithms were selected for evaluation. Among them, Radix-2 FFT, FIR filtering, and 2D convolution were identified as compute-bound due to their high data reuse, especially when supported by the cache hierarchy. In contrast, the sorting algorithms exhibited memory-bound behavior, primarily due to frequent element swapping that limits effective data reuse.

The compute-bound algorithms benefited significantly from both temporal and spatial locality, leading to better speedup compared to their memory-bound counterparts. Additionally, these algorithms are naturally well-suited to parallel execution. For instance, once the FIR filter has loaded its input data and coefficients into the local caches, it can proceed with minimal core-to-core synchronization, enabling smooth and efficient parallel processing.

Interestingly, bubble sort, despite being a memory-bound algorithm, achieved the highest observed speedup—nearly 3× faster than its single-core counterpart. This result is somewhat counterintuitive, as the algorithm involves frequent data swapping, which typically triggers cache invalidations and incurs coherence overhead due to the protocol in place. However, several factors may contribute to this unexpected performance gain.

First, the simplicity and regularity of bubble sort's memory access pattern may allow the cache system to still exploit some degree of spatial locality, especially with particular datasets. Second, the algorithm benefits greatly from fine-grained parallelization—by splitting the workload and assigning independent sections of the array to each core, the overhead from inter-core synchronization is minimized. Lastly, load balancing is naturally well-distributed in bubble sort's inner loop, which ensures both cores remain active for a substantial portion of the runtime. In contrast, the quicksort algorithm, when evaluated under its worst-case input conditions, demonstrates a slight performance degradation, achieving only 0.75× the performance of the single-core implementation. However, under best-case conditions, the performance closely aligns with that of the single-core baseline. This behavior can be attributed to the recursive nature of the algorithm, which generates a highly dynamic control flow, creating nonsequential memory accesses, as well as the frequent cache invalidations caused by continual swapping of data elements. These factors lead to reduced cache efficiency and increased memory traffic, limiting the benefits of parallel execution.

This also demonstrates that, across the majority of the selected programs, the dual-core processing system outperformed its 50 MHz single-core counterpart [9], despite operating at a reduced frequency of 30 MHz. This highlights the performance gains achievable through parallelism, even under lower clock frequencies. On average, the system achieves a speedup of 1.64× compared to its single-core counterpart without caches. While 4 out of 6 benchmarks achieved notable speedups in the dual-core design, particularly the compute-bound kernels and bubble sort, the remaining 2 (quicksort in both best and worst case) experienced minor slowdowns due to poor cache behavior and frequent invalidations. Nonetheless, the high gains from the faster-executing workloads outweighed the losses, resulting in an overall average speedup of 1.64×.

Finally, the implemented design shows a Worst Negative Slack of 0.921 ns. With an operating frequency of 30 MHz, the maximum allowable frequency is therefore 30.852 MHz.

V. CONCLUSION AND RECOMMENDATIONS

This work has demonstrated the viability of implementing a general-purpose dual-core RISC-V processor system with integrated L1 and L2 data caches on an FPGA platform, serving as an initial step toward scalable multicore architectures for wireless sensor nodes. By extending an existing RISC-V core and incorporating dedicated local memory per core, we enabled parallel execution within a shared memory system.

Despite operating at a reduced frequency, the dual-core system achieved an average speedup of 1.64× across the evaluated programs. This highlights the effectiveness of parallel execution in improving throughput, even under tighter timing constraints and lower operating frequencies.

The lightweight nature of the design—relative to the capacity of the XC7A200T FPGA—ensures significant headroom for further expansion, whether by scaling computational resources, increasing cache capacity, or integrating additional system features. The design scales efficiently, requiring at most 1.90× the resource utilization of the base single-core processor while delivering significant performance improvements—between 1.49× and 2.90× speedup—across diverse workloads.

Finally, we recommend exploring advanced architectural techniques to enable parallel access to shared components, particularly the L2 cache. Allowing concurrent access from multiple cores can significantly improve system efficiency and overall throughput by minimizing contention and reducing memory access latency. Such enhancements are essential for scaling the architecture to support more cores and higher-performance workloads.

References:

[1] Munir A, Gordon-Ross A, Ranka S. 2014. Multi-core embedded wireless sensor networks: architecture and applications. IEEE Transactions on Parallel and Distributed Systems. 25(6):1553–1562. https://doi.org/10.1109/TPDS.2013.219

[2] Jang H et al. 2021. Developing a multicore platform utilizing open RISC-V cores. IEEE Access. 9:120 010–120 023). https://doi.org/10.1109/ACCESS.2021.3108475.

[3] Kumar MA, Francis. 2017. Survey on various advanced techniques for cache optimization methods for RISC-based system architecture. 4th International Conference on Electronics and Communication Systems (ICECS). p. 195–200. https://doi.org/10.1109/ECS.2017.8067868.

[4] Cui E, Li T, Wei Q. 2023. RISC-V instruction set architecture extensions: a survey. IEEE Access.11: 24 696–24 711. https://doi.org/10.1109/ACCESS.2023.3246491.

[5] Kamaleldin A, Hesham S, Gohringer D. 2020. Towards a modular RISC-V-based many-core architecture for FPGA accelerators. IEEE Access. 8:148812-1488262020. https://doi.org/ 10.1109/ACCESS.2020.3015706.

[6] Pullini A, Rossi D, Loi I, Di Mauro A, Benini L. 2018. Mr. Wolf: A 1 GFLOP/s energy-proportional parallel ultra low power SOC for IOT edge processing. in ESSCIRC. IEEE 44th European Solid State Circuits Conference (ESSCIRC). pp. 274–277. https://doi.org/10.1109/ESSSCIRC.2018.8494247.

[7] Dogan AY, Atienza D, Burg A, Loi I, Benini L. 2011. Power/performance exploration of single-core and multi-core processor approaches for biomedical signal processing. Integrated Circuit and System Design. Power and Timing Modeling, Optimization, and Simulation. Berlin, Heidelberg: Springer. p. 102–111.

[8] Alidio L, Fernandez JL, Montes JAG, Palma CM. 2009. Implementation of a 32-Bit dual core ARM7 microprocessor with split-private level one cache and shared level two data cache [Undergraduate]. University of the Philippines Diliman.

[9] Neri MJ, RI Ridao, Baylosis VE, Chua, PM, Tan, AJ, de Leon, MT. 2020. Design and implementation of a pipelined RV32IMC processor with interrupt support for large-scale wireless sensor networks. IEEE Region 10 Conference (TENCON). p. 806–811. https://doi.org/10.1109/TENCON50793.2020.9293798

[10] Papamarcos MS, Patel JH. 1984. A low-overhead coherence solution for multiprocessors with private cache memories. 11th Annual International Symposium on Computer Architecture (ISCA '84). Association for Computing Machinery; New York, USA. p. 348–354. https://doi.org/10.1145/800015.808204