



SoC-FPGA Ortamında AXI-DMA ve BRAM Tabanlı Entegrasyon Yöntemlerinde Gecikme ve Kaynak Tüketim Analizi

Güner TATAR^{1*}  

¹Elektrik-Elektronik Mühendisliği, Mühendislik ve Doğa Bilimleri Fakültesi, İstanbul Medeniyet Üniversitesi, İstanbul, Türkiye.

¹guner.tatar@medeniyet.edu.tr

Geliş Tarihi: 23.9.2025
Kabul Tarihi: 22.1.2026

Düzelme Tarihi: 12.1.2026

doi: <https://doi.org/10.62520/fujece.1790038>
Araştırma Makalesi

Alıntı: G. Tatar, "SoC-FPGA ortamında AXI-DMA ve BRAM tabanlı entegrasyon yöntemlerinde gecikme ve kaynak tüketim analizi" Fırat Üni. Deny. ve Hes. Müh. Derg., vol. 5, no 1, pp. 316-329, Şubat 2026.

Özet

Bu makale, Xilinx Zynq-7000 SoC üzerinde tasarlanan iki entegrasyon stratejisinin karşılaştırmalı değerlendirmesini sunmaktadır. Araştırmada, Advanced eXtensible Interface Direct Memory Access (AXI-DMA) tabanlı mimari ile Block Read Access Memory (BRAM) tabanlı mimari, performans, kaynak kullanımı ve gecikme açısından ele alınmıştır. Her iki tasarım da aritmetik işlemler için özel olarak geliştirilmiş bir işlem elemanı içermektedir; ancak veri aktarımı ve arabellekleme mekanizmaları bakımından önemli farklılıklar göstermektedir. AXI-DMA tabanlı tasarımda, işlemci sistemi ile programlanabilir mantık (PL) arasındaki iletişim, bir DMA motoru tarafından kontrol edilen AXI4-Stream arayüzü üzerinden sağlanmaktadır. BRAM tabanlı mimaride ise AXI BRAM denetleyicileri aracılığıyla erişilen çift portlu blok bellekler kullanılmakta ve doğrudan veri erişimi mümkün olmaktadır. Uygulama sonuçları, her iki mimarinin de XC7Z020 cihazının kaynak kısıtlarını karşıladığını göstermektedir. Bununla birlikte, AXI-DMA tabanlı tasarımın donanım kaynak tüketiminin, BRAM tabanlı mimariye kıyasla ortalama %54 daha yüksek olduğu belirlenmiştir. Performans analizleri ayrıca gecikme açısından da çarpıcı bir farklılığı ortaya koymaktadır. AXI-DMA mimarisinde işlem başına ortalama 1,19 ms süre gerekmektedirken, BRAM tabanlı yaklaşım yaklaşık 0,10 ms daha düşük gecikme sağlamış ve toplam yürütme süresini 359.919 µs'den 32.487 µs'ye düşürmüştür. Elde edilen bulgular, ölçeklenebilirlik ile gecikme arasında belirgin bir ödünleşim bulunduğunu ortaya koymaktadır. AXI-DMA mimarisi, akış odaklı uygulamalarda esneklik ve yüksek veri aktarım kapasitesi sunarken, BRAM tabanlı mimari daha küçük ölçekli ve düşük gecikme gerektiren senaryolarda daha yüksek verimlilik sağlamaktadır. Çalışma, heterojen bilgi işlem platformlarında Field Programmable Gate Array (FPGA) tabanlı hızlandırıcıların tasarımı için yol gösterici nitelikte sonuçlar sunmaktadır.

Anahtar kelimeler: SoC-FPGA uyum stratejisi, AXI-DMA mimarisi, BRAM tabanlı tasarım, Kaynak kullanımı ve gecikme analizi, FPGA tabanlı heterojen hesaplama

*Yazışılan yazar



Latency and Resource Trade-off Analysis of AXI-DMA and BRAM Integration Approaches on SoC-FPGA

Güner TATAR^{1*}  

¹Electrical-Electronics Engineering, Faculty of Engineering and Natural Sciences, Istanbul Medeniyet University, Istanbul, Türkiye.

¹guner.tatar@medeniyet.edu.tr

Received: 23.9.2025

Accepted: 22.1.2026

Revision: 12.1.2026

doi: <https://doi.org/10.62520/fujece.1790038>

Research Article

Citation: G. Tatar, "Latency and resource trade-off analysis of AXI-DMA and BRAM integration approaches on SoC-FPGAs", *Firat Univ. Jour. of Exper. and Comp. Eng.*, vol. 5, no 1, pp. 316-329, February 2026.

Abstract

This paper presents a comparative evaluation of two integration strategies for the Xilinx Zynq-7000 System-on-Chip (SoC): an Advanced eXtensible Interface-Direct Memory Access (AXI-DMA)-based architecture and a Block RAM (BRAM)-based architecture. Both designs employ a custom processing element (PE) for arithmetic operations, yet they differ significantly in data transfer and buffering mechanisms. In the AXI DMA design, communication between the processing system (PS) and programmable logic (PL) is achieved via an AXI4-Stream interface controlled by a DMA engine. In contrast, the BRAM-based design uses dual-port block memories via AXI BRAM controllers, enabling direct operand access. Implementation results indicate that both designs comfortably meet the resource constraints of the XC7Z020 device. However, the AXI DMA-based architecture exhibits higher hardware resource utilization, with average consumption approximately 54% greater than that of the BRAM-based design. Performance analysis reveals a pronounced latency difference: the AXI DMA design required an average of ~1.19 ms per operation. In comparison, the BRAM-based approach achieved a reduction of ~0.10 ms, resulting in a total execution time of 32.487 μ s compared to 359.919 μ s. These findings demonstrate a clear trade-off between scalability and latency. While AXI DMA provides flexibility and throughput for stream-oriented applications, BRAM-based integration delivers superior efficiency in small-scale, low-latency scenarios. The study offers practical insights for guiding the design of Field-Programmable Gate Array (FPGA)-based accelerators on heterogeneous computing platforms.

Keywords: SoC-FPGA integration strategy, AXI-DMA architecture, BRAM-based design, Resource utilization and latency analysis, FPGA-based heterogeneous computing

*Corresponding author

1. Introduction

FPGAs are widely used to accelerate applications that require high computational performance, low power consumption, and design flexibility [1–3]. Unlike general-purpose processors, FPGAs allow designers to implement application-specific hardware pipelines, making them highly efficient for workloads such as digital signal processing, image processing, and cryptography [2–4]. With the emergence of heterogeneous SoC platforms, such as the Xilinx Zynq-7000 family, an ARM-based PS and PL can be integrated on the same chip, enabling tight coupling between software and hardware accelerators [1].

A critical challenge in such SoC-FPGA platforms is the efficient transfer of data between the PS and the PL. The communication mechanism chosen for this interaction has a dominant impact on overall system performance, particularly in terms of latency and resource usage [5-8]. Two widely used integration strategies are AXI-DMA and BRAM-based memory-mapped communication. AXI-DMA provides a scalable, high-throughput streaming interface well-suited for data-intensive, continuous workloads. In contrast, BRAM-based integration relies on on-chip block memories, enabling fast and deterministic access with very low latency, making it attractive for small or control-oriented workloads [6,9-11].

Although both approaches are commonly used, most existing studies focus on either DMA-based streaming accelerators or BRAM-based memory-centric designs in isolation. Works such as [1,5], and [11] demonstrate the advantages of AXI-DMA for high-throughput and large-scale workloads, while studies such as [6] and [13] emphasize the low-latency benefits of BRAM-based architectures. However, these studies do not provide a controlled and fair comparison of AXI-DMA and BRAM-based integration implemented on the same SoC platform using the same processing core. As a result, the practical trade-offs among latency, resource utilization, and architectural overhead remain insufficiently quantified.

This paper addresses this gap by implementing both AXI-DMA-based and BRAM-based integration strategies on the same Xilinx Zynq-7000 SoC using an identical custom arithmetic processing element. By keeping the computation kernel, clock frequency, and experimental conditions fixed, the impact of communication and buffering mechanisms can be isolated and evaluated in a fair, reproducible manner.

1.1. Contribution

The main contributions of this work are summarized as follows:

1. A fair and controlled comparison of AXI-DMA and BRAM-based integration strategies on the same Zynq-7000 SoC using an identical custom arithmetic processing element, eliminating architectural and computational bias.
2. A quantitative evaluation of resource utilization and latency, showing that the BRAM-based design achieves more than an order-of-magnitude reduction in execution latency while consuming significantly fewer hardware resources than the AXI-DMA-based approach.
3. A systematic analysis of the scalability–latency trade-off between streaming-based (AXI-DMA) and memory-mapped (BRAM) accelerator integration, providing practical design guidelines for selecting the appropriate strategy depending on whether an application is throughput-oriented or latency-critical.

2. Background and Related Works

FPGAs have been widely adopted as acceleration platforms for applications requiring high parallelism, low latency, and real-time responsiveness. Their reconfigurable nature and ability to tightly integrate custom hardware with general-purpose processors make FPGA-based SoCs particularly attractive for embedded and heterogeneous computing. In our earlier works [2,4], we demonstrated how reconfigurable MPSoC-FPGA architectures can be exploited for multi-task and multi-learning ADAS, highlighting the effectiveness of pipelined hardware acceleration in safety-critical automotive applications. These studies emphasized the strengths of FPGA-based platforms in achieving both high computational throughput and deterministic real-

time behavior. In contrast, the present work targets more fundamental accelerator kernels. It focuses explicitly on how data transfer mechanisms between the PS and PL influence latency and hardware resource utilization.

Beyond automotive applications, heterogeneous FPGA-based SoC platforms such as the Xilinx Zynq-7000 family have been extensively explored across a wide range of domains. Rios-Navarro et al., [1] investigated hardware/software co-design strategies for CNN accelerators and provided a detailed analysis of memory transfers in SoC-based systems. Their results underline that PS–PL communication efficiency plays a critical role in determining overall system performance and achievable speedup. Although their work focuses on CNN workloads, it demonstrates that communication overhead can significantly limit acceleration gains, motivating more focused investigations into PS–PL data transfer mechanisms independent of application complexity.

When considering communication methods, several studies have highlighted the scalability and throughput advantages of AXI-based and DMA-driven data transfer. Wang et al. [5,12] analyzed scatter–gather DMA performance on SoC-FPGA platforms and showed that AXI-DMA can sustain high throughput for extensive and continuous data streams. Similarly, Canto-Navarro et al. [11] designed an AXI-based FPGA accelerator for cryptographic workloads, demonstrating the effectiveness of AXI-Stream interfaces in bandwidth-intensive applications. However, these studies primarily emphasize throughput and scalability, while per-transaction latency and control overhead receive less attention. As a result, the suitability of AXI-DMA for short, iterative, or latency-critical workloads remains insufficiently explored.

The impact of DMA-induced overhead is further highlighted by Parameshwara et al. [14], who presented an FPGA-accelerated RISC-V architecture with custom ISA extensions for neural network inference on edge devices. Their real-hardware evaluation on a Zynq-based SoC demonstrates that, despite significant computational acceleration, memory transfers and DMA-based communication account for a substantial portion of total execution time. The authors show that data movement overhead can limit achievable speedup, emphasizing that PS–PL communication efficiency is often as critical as the accelerator design itself. While their work focuses on neural network acceleration at the ISA level, it strongly motivates a deeper examination of DMA-related latency in SoC-FPGA systems.

In contrast to DMA-centric approaches, BRAM-based and memory-mapped integration strategies have been widely adopted in latency-sensitive scenarios. Cilasun [6] modeled FPGA memory systems with variable latency and demonstrated that on-chip BRAM access can significantly reduce communication overhead. Likewise, Kieu-Do-Nguyen et al. [13] proposed a low-latency FPGA architecture for post-quantum cryptographic applications, highlighting the benefits of tightly coupled on-chip memory structures. These studies confirm the latency advantages of BRAM-based designs; however, they do not provide a direct comparison with AXI-DMA-based integration under identical hardware and experimental conditions.

A more communication-centric perspective is provided by Irtija et al. [15], who conducted a detailed analysis of digital communication mechanisms in SoC-based systems, focusing on latency, throughput, and reliability of data transfer between processing and programmable logic. Their study evaluates multiple communication approaches, including memory-mapped interfaces and DMA-based transfers, and shows that communication overhead can dominate system-level performance. Although their application domain targets quantum control systems, the reported latency measurements and communication analysis are directly applicable to general-purpose SoC-FPGA platforms.

A complementary, accelerator-oriented study is presented by Axinte et al. [16], who conducted a comprehensive evaluation of embedded streaming hardware-accelerator interconnect architectures on FPGA-based SoCs. Their work compares tightly coupled streaming, FIFO-based adapters, and AXI-DMA-driven architectures, with a particular focus on communication latency and FPGA resource utilization. Experimental results show that while DMA-based streaming can reduce frame-level latency in well-organized streaming scenarios, it incurs substantial hardware overhead and is highly sensitive to data organization and buffer descriptor management. Notably, the authors report that suboptimal DMA data organization can result in a more than 7-fold increase in latency for identical data sizes, highlighting the

inherent trade-off among scalability, flexibility, and latency in AXI-DMA-based communication architectures.

While the aforementioned studies provide valuable insights into PS–PL communication and accelerator integration, most existing work either focuses on application-specific designs or emphasizes throughput-oriented streaming architectures. Direct, fair comparisons between AXI-DMA-based streaming and BRAM-based memory-mapped integration, using identical processing elements and experimental conditions, remain limited in the literature. Addressing this gap, the present work isolates the impact of communication and buffering mechanisms by evaluating both integration strategies on the same Zynq-7000 SoC, with identical clock frequency, processing element, and workload characteristics. This controlled approach enables an application-independent assessment of latency–resource trade-offs and provides practical design guidelines for selecting PS–PL integration strategies based on whether an application prioritizes throughput or low-latency response.

Finally, Berrazueta-Mena and Navas [17] evaluated multiple compute-intensive hardware accelerators on Zynq-based SoCs using an HLS-driven design flow, highlighting performance–resource trade-offs in heterogeneous FPGA-based systems. Their work demonstrates the effectiveness of AXI-based accelerator integration and provides a comprehensive benchmarking framework for Zynq platforms. However, their study relies primarily on AXI-based communication and does not explicitly compare DMA-driven streaming with BRAM-based memory-mapped integration under identical conditions. The present work complements this limitation by focusing explicitly on PS–PL communication mechanisms and their impact on latency and resource utilization, thereby extending existing benchmarking-oriented studies toward communication-aware design analysis.

3. System Design and Methodology

3.1. Overall design flow

The study was conducted on the Xilinx Zynq-7000 SoC platform, utilizing both the PS and PL to implement and test the hardware accelerators. A custom PE was designed to perform basic arithmetic operations (addition, subtraction, and multiplication). The primary distinction between the two approaches lies in how data is transferred and buffered between the PS and PL.

In the BRAM-based design, shown in Figure 1, data exchange between the PS and PL is managed through dual-port block memories accessed via AXI BRAM controllers. The design begins with a BRAM A input, which stores 32-bit data values. These inputs are transferred to the input registers, where operands a , b , and y_in are organized and aligned for arithmetic operations. The PE, specifically developed to support addition, subtraction, and multiply–accumulate operations, performs the core computations. A controller based on a finite state machine (FSM) coordinates the overall flow by generating addresses, managing counters, and signaling data readiness. To guarantee proper synchronization, a start-edge detection module ensures accurate triggering through rising-edge signals. After computation, results are stored in the output register in signed 8-bit format. A sign-extension stage then scales the results to 32-bit to match the BRAM data width, after which the processed values are written into the BRAM output module for efficient storage and later retrieval. This modular organization demonstrates the efficiency of the BRAM-based approach, which minimizes latency by enabling direct operand access while preserving low control complexity.

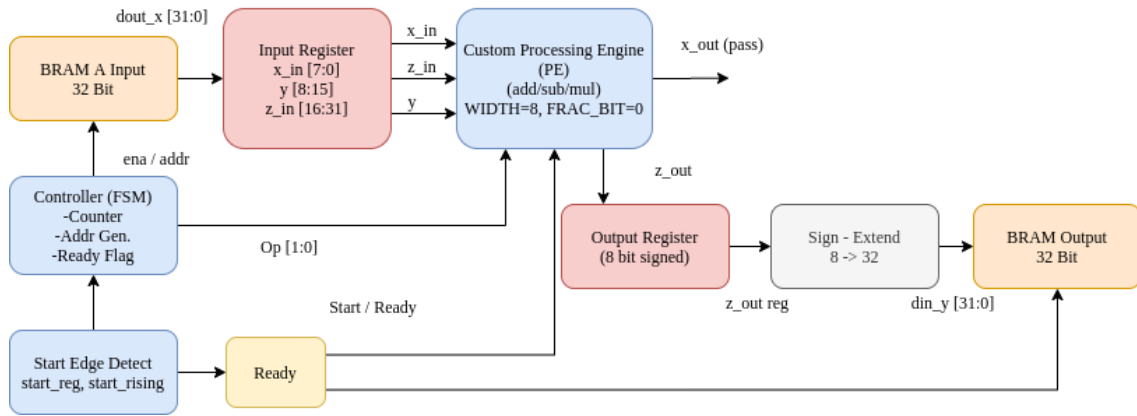


Figure 1. BRAM-based block design for data transfer and buffering between PS and PE

By contrast, the AXI-DMA-based design, illustrated in Figure 2, utilizes direct memory access to transfer data between the PS and PL. In this architecture, the PE is wrapped in an AXI-Stream AXIS interface, where inputs x_{in} , y , and z_{in} are received via the slave port and the computed result z_{out} is sent through the master port. The AXI DMA IP handles conversion between memory-mapped transactions (in DDR or other PS memory) and streaming transactions in the PL.

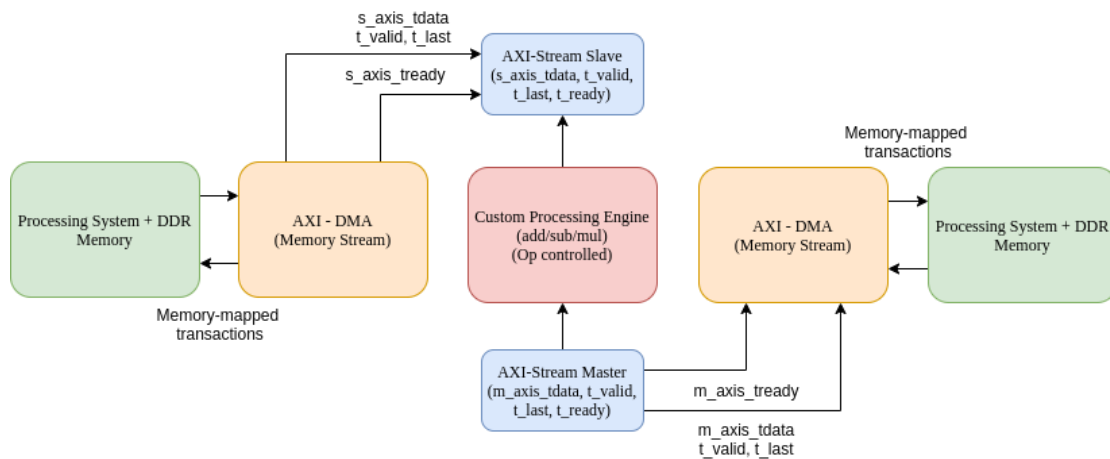


Figure 2. AXI-DMA-based integration architecture with AXI-Stream slave and master interfaces for data transfer between PS and PE

Control and data signals such as $tdata$, $tready$, $tvalid$, and $tlast$ participate in protocol handshaking to ensure correct streaming operation. The PS initiates transfers via the AXI DMA, specifying source and destination addresses; DMA moves chunks of data without CPU intervention. This design introduces additional control logic compared to the BRAM-based approach but offers higher throughput and scalability, especially for continuous or burst data streaming.

3.2. AXI-DMA based architecture

In the AXI DMA-based design, data transfer between the PS and the PL is handled through the AXI4-Stream interface, coordinated by the DMA engine. This integration allows operands stored in DDR memory to be efficiently streamed into the custom PE, with results written back to memory via the same mechanism. The PE is implemented as a custom RTL VHDL module, synthesized separately, and packaged as an IP core before being integrated into the block design. The system was created in Vivado Design Suite 2022.1, leveraging the IP Integrator for block-level composition. The Zynq-7000 Processing System block provides the ARM Cortex-A9 subsystem with DDR and peripheral interfaces. An AXI DMA IP core is connected through an AXI Interconnect to enable memory-mapped transactions between DDR and the accelerator. The

axis_pe_0 block, representing the custom PE, is connected to the streaming ports of the DMA (M_AXIS_MM2S for data input and S_AXIS_S2MM for data output). Additionally, AXI GPIO modules were included to configure operation modes (addition, subtraction, multiplication) and control accelerator execution. Reset and clock signals are distributed via the Processor System Reset and FCLK outputs of the PS. The resulting block design, shown in Figure 3, demonstrates the complete AXI DMA-based integration. Here, the PS communicates with the DMA through GP master ports, while high-performance (HP) ports are employed for data movement. The interconnect fabric handles arbitration between master and slave interfaces, ensuring seamless communication between the PS memory subsystem and the accelerator.

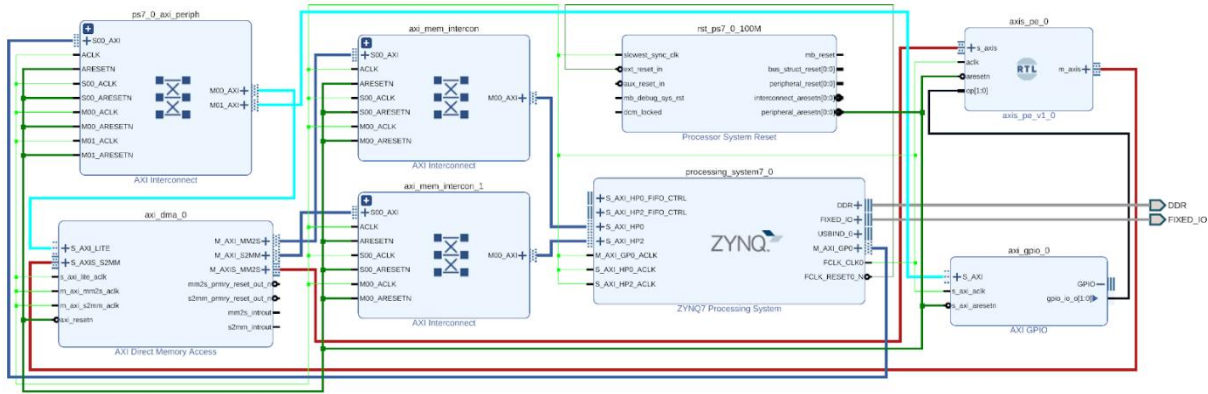


Figure 3. Custom block design of the AXI DMA-based architecture

3.3. BRAM based architecture

The BRAM-based design utilizes dual-port BRAMs interconnected via AXI BRAM controllers. This setup enables direct memory-mapped access for reading and writing operands. The PE is connected directly to BRAMs, eliminating the overhead of streaming interfaces. This architecture is particularly effective for applications where low latency is critical and data size is relatively small. As illustrated in Figure 4, the BRAM-based design enables direct operand access through on-chip block memories with minimal interface complexity.

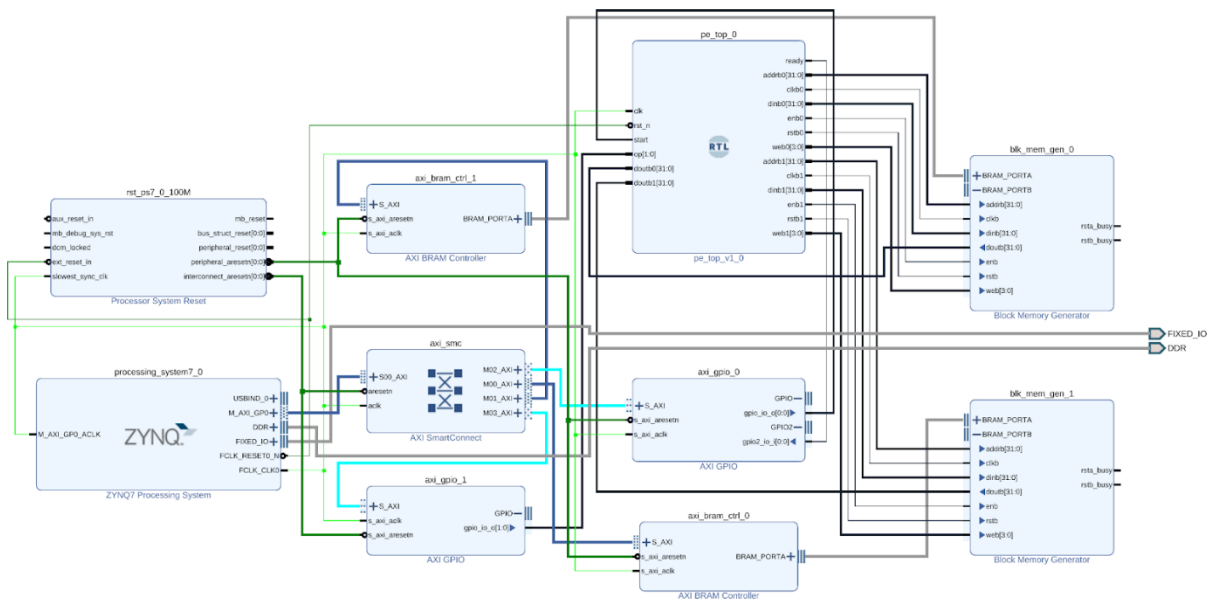


Figure 4. Custom block design of the BRAM-based architecture

3.4. Processing engine (PE)

In both architectures, the core computational unit is a custom PE implemented in VHDL and integrated into the PL. The PE supports three arithmetic operations: addition, subtraction, and multiplication. The desired operation is selected through a 2-bit control signal (op), while two 32-bit operands are provided as inputs. The computed result is then delivered to the output interface, which connects either to the AXI-DMA stream or to the BRAM controller, depending on the architecture. As illustrated in Figure 5, the PE design consists of two operand inputs, an operation selector, and a single result output. The simplicity of this structure ensures minimal hardware overhead while maintaining flexibility in executing different arithmetic tasks.

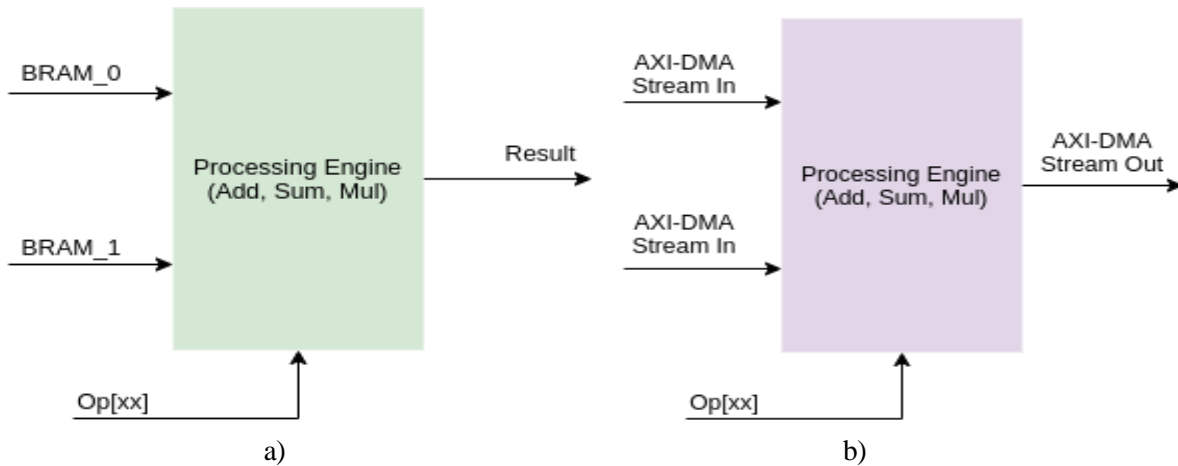


Figure 5. PE structure. a) BRAM-Based PE, b) AXI-DMA-based PE

4. Implementation Details

4.1. Development environment

Both architectures were implemented and tested on the Xilinx Pynq-Z1 development board, which is based on the Zynq-7000 SoC (XC7Z020-1CLG400C). The designs were created using the Vivado Design Suite, with block diagrams generated through the IP integrator. The PE was developed in HDL and integrated as a custom IP core. For software-side control, C/C++ programs were executed on the ARM Cortex-A9 processor within the PS.

4.2. Hardware design integration

The two integration strategies were realized as complete hardware systems. In the AXI DMA-based design, the DMA engine was configured to stream data between DDR memory and the PE via AXI4-Stream connections, as shown in Figure 1. In the BRAM-based design, dual-port block memories were connected to the PE through AXI BRAM controllers, allowing direct operand access, as illustrated in Figure 2. Both systems included GPIO modules for basic control and status monitoring.

4.3. PE implementation

The PE, illustrated in Figure 5, was designed to provide basic arithmetic functionality with minimal hardware overhead while adapting to two different integration strategies. In both cases, the PE accepts 32-bit input words containing operands a and b , together with an operation code that selects addition, subtraction, or multiplication. Computation is performed in a single clock cycle, and results are written back to the system through either block memories or streaming interfaces, depending on the surrounding architecture.

Algorithm 1: Pseudo-code for BRAM-based PE

```

initialize FSM, counters, and BRAM addresses
loop:
  read operand_a, operand_b, and y_in from BRAM input registers
  latch operands into PE registers
  case (op):
    00 -> result := operand_a + operand_b
    01 -> result := operand_a - operand_b
    10 -> result := operand_a * operand_b
  end case
  extend result from 8 bits to 32 bits
  write result to BRAM output register
  update FSM and counters
end loop

```

In the BRAM-based design, operands are fetched from dual-port block RAMs via AXI BRAM controllers. A finite-state machine supervises address generation, counters, and read/write sequencing, ensuring that operands are correctly aligned before being latched into the datapath. The step-by-step execution of this process is outlined in *Pseudo-code 1*. Arithmetic results are produced in 8-bit format, extended to 32 bits for consistency with the BRAM interface, and written back to memory through output registers. This scheme achieves deterministic operation with low latency by enabling direct operand access without streaming overhead.

Algorithm 2: Pseudo-code for AXI-DMA-based PE

```

loop:
  wait until s_axis_tvalid = '1' and s_axis_tready = '1'
  latch input_word from s_axis_tdata
  extract operand_a, operand_b, and op from input_word
  case (op):
    00 -> result := operand_a + operand_b
    01 -> result := operand_a - operand_b
    10 -> result := operand_a * operand_b
  end case
  store result in result_reg
  set m_axis_tdata := result_reg
  assert m_axis_tvalid
  if m_axis_tready = '1':
    send result word to DMA
  deassert m_axis_tvalid
end loop

```

In the AXI-DMA-based integration, data are transferred between PS memory and the PE through AXI DMA, which converts memory-mapped transactions into AXI4-Stream transfers. Input words arrive at the s_axis interface along with handshake signals (tvalid, tready, tlast), and the corresponding execution steps are summarized in *Pseudo-code 2*. The PE extracts operands from the input word, executes the specified operation in one cycle, and forwards the result through the m_axis interface. Output handshake signals ensure that results are delivered only when the downstream path is ready, while a lightweight buffer prevents data loss during backpressure. This design supports high-throughput streaming with near one-word-per-cycle performance under ideal conditions.

Through these two integration strategies, the same arithmetic kernel can operate either as a memory-centric accelerator or as a stream-oriented processing element. While the BRAM-based organization emphasizes direct access and minimal latency, the AXI-DMA-based version enhances scalability and throughput, thus demonstrating the flexibility of the PE within heterogeneous SoC-FPGA platforms.

4.4. Resource utilization

After the implementation, resource utilization reports were examined for both architectures. Each design fits well within the available resources of the XC7Z020 device. The AXI DMA-based system, however, showed a modest increase in utilization, mainly due to the added interconnect and DMA control logic. This overhead was reflected in higher LUT, FF, and BRAM usage when compared with the BRAM-based alternative. DSP consumption remained identical across both implementations, as the same arithmetic unit was employed in each case. Table 1 summarizes these results, making clear the additional cost of integrating AXI DMA.

Table 1. Resource utilization comparison between AXI DMA and BRAM-based designs

Resource	Available	AXI-DMA based Utilization	Utilization (%)	BRAM-based Utilization	Utilization (%)
LUT	53200	5804	10.91	2354	4.42
LutRAM	17400	861	4.95	202	1.16
FFs	106400	5950	5.59	3186	2.99
BRAM	140	4	2.86	2	1.43
BUFG	32	1	3.13	1	3.13
Power (W)	-	1.466	-	1.408	-

4.5. Experimental setup

The experimental evaluation was carried out on a Pynq-Z1 development board, which integrates a Xilinx Zynq-7000 SoC (XC7Z020) comprising a dual-core ARM Cortex-A9 PS and a PL. The board was connected to a host computer via USB and Ethernet interfaces for programming, configuration, and data exchange. The PS executed C/C++ control software that managed data transfers, triggered the accelerators, and measured execution times.

For both AXI-DMA and BRAM-based designs, identical test conditions were applied to ensure a fair comparison. Input operands were generated and stored in PS memory, and the accelerator was invoked through memory-mapped control registers. In the AXI-DMA-based system, data blocks were transferred between DDR memory and the PE using the AXI DMA engine, while in the BRAM-based system, operands and results were accessed directly through dual-port BRAMs via AXI BRAM controllers.

Each experiment consisted of 100 consecutive iterations of arithmetic operations, including addition, subtraction, and multiplication. The ARM Cortex-A9 timer registers were used to measure the execution time from the moment the accelerator was triggered until the final result was written back to memory. To minimize measurement noise, the results were averaged over all iterations.

Figure 6 illustrates the physical experimental setup, showing the Pynq-Z1 board connected to the host computer. This setup demonstrates the complete hardware–software co-design environment, in which the PS manages data movement and control. At the same time, the PL executes the hardware-accelerated arithmetic operations in real time.

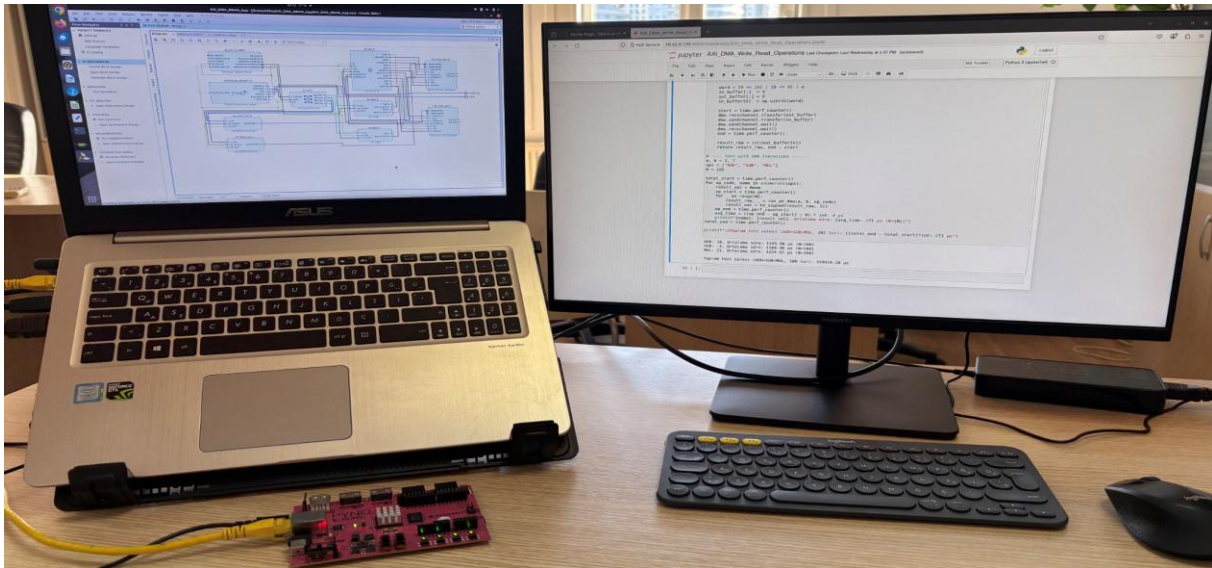


Figure 6. Experimental setup showing the Pynq-Z1 development board connected to the host computer via USB and Ethernet. The ARM-based PS executes the control software and manages data transfers. At the same time, the PL runs the hardware accelerator using either AXI-DMA or BRAM-based integration

5. Performance Evaluation

The performance of both architectures was evaluated on the Pynq-Z1 development board using 100 consecutive iterations of arithmetic operations. The test program, executed on the ARM Cortex-A9 processing system, initialized the operands, triggered the accelerator, and collected timing results. Latency measurements were obtained using ARM-side cycle counters, capturing the total time elapsed from accelerator invocation to result availability in memory. All reported results represent average values over the 100 iterations, reducing the impact of transient effects and ensuring statistically meaningful and reliable performance comparison.

This evaluation methodology allows the communication and control overhead between the PS and PL to be directly reflected in the measured latency, making it well suited for comparing different integration strategies under identical experimental conditions.

5.1. Average latency results

The measurements reveal a clear and consistent difference in execution time between the two integration strategies. In the AXI-DMA-based design, the average operation time across all three arithmetic functions (addition, subtraction, and multiplication) was approximately 1.19 ms per iteration. In contrast, the BRAM-based design achieved an average latency of approximately 0.10 ms per iteration.

Table 2 summarizes the average latency per operation and the total execution time over 100 iterations for both architectures. The results indicate that the BRAM-based approach reduces the total execution time from 359,919 μ s to 32,487 μ s, corresponding to more than an order-of-magnitude improvement in latency.

Table 2. Average latency and total execution time over 100 iterations for AXI-DMA and BRAM-based designs

Metric	AXI-DMA Design	BRAM based Design
Average latency per operation (μ s)	~1190.97	~99.79
Total execution time (100 iterations, μ s)	359,919.28	32,486.85

To better illustrate this performance gap, Figure 7 presents the experimental results in two forms: (a) average latency per operation and (b) total execution time over 100 iterations. The results clearly demonstrate that

BRAM-based integration significantly outperforms the AXI-DMA-based approach in latency-critical scenarios.

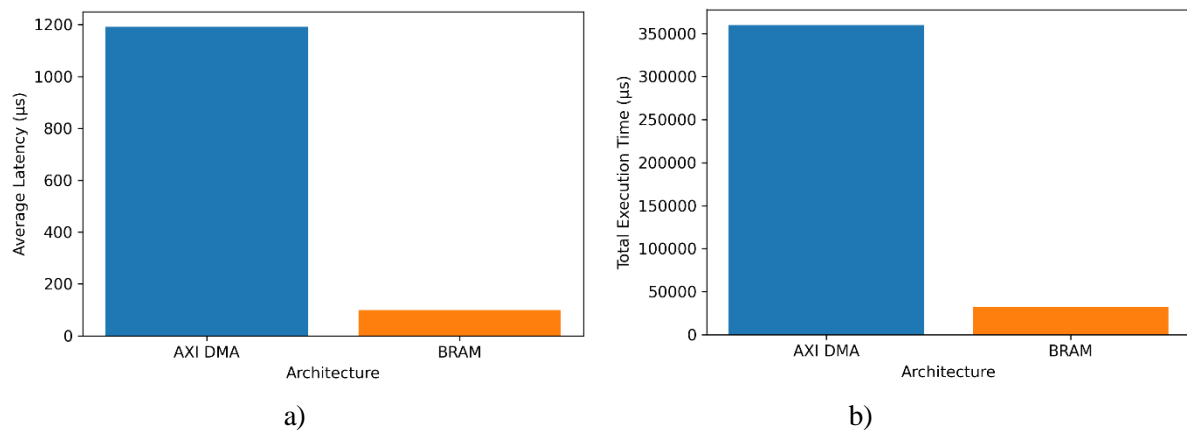


Figure 7. Performance comparison between AXI-DMA and BRAM-based designs a) Average latency per operation, b) Total execution time for 100 iterations

6. Discussion of results

The experimental results highlight a fundamental trade-off between AXI-DMA-based and BRAM-based integration strategies in SoC-FPGA platforms. Although both architectures employ the same processing element and operate at the same clock frequency and under identical experimental conditions, their performance characteristics differ substantially due to their data transfer and buffering mechanisms.

The AXI-DMA-based design introduces noticeable communication overhead, resulting in an average latency of approximately 1.19 ms per operation. This overhead arises from several factors, including DMA configuration, memory-to-stream and stream-to-memory transactions, buffer descriptor handling, and AXI protocol handshaking. While these mechanisms enable high throughput and scalability for large data blocks and continuous streaming workloads, they introduce a fixed per-transaction cost that becomes dominant in short, iterative, or latency-sensitive computations.

In contrast, the BRAM-based design achieves a much lower average latency of approximately 0.10 ms per operation. This more than tenfold reduction is primarily attributed to the use of on-chip block memories and direct memory-mapped access, which eliminates the need for DMA setup and streaming handshakes. By allowing the processing element to access operands and store results directly in BRAM, the communication path becomes shorter, simpler, and more deterministic, leading to reduced execution time and lower latency variability.

Resource utilization results further reinforce this observation. The AXI-DMA-based architecture consumes significantly more LUTs, flip-flops, and BRAM resources due to the inclusion of the DMA engine, AXI interconnects, and streaming interfaces. Although this additional hardware provides flexibility and scalability, it increases design complexity and hardware cost. In contrast, the BRAM-based design minimizes control logic and interface overhead, resulting in both lower latency and reduced resource consumption.

Overall, the results indicate that AXI-DMA-based integration is well-suited for throughput-oriented and continuous data-streaming applications, where the communication overhead can be amortized across large data transfers. Conversely, BRAM-based integration is more efficient for small-scale, low-latency, and iterative workloads, where fast response time and deterministic behavior are critical. Therefore, the selection of the integration strategy should be guided by application requirements, particularly the trade-off between throughput scalability and latency sensitivity.

7. Conclusion and Future Work

This paper presented a comparative study of AXI DMA- and BRAM-based accelerator integration on the Xilinx Zynq-7000 SoC. A custom processing element was implemented in both architectures, and the designs were evaluated in terms of resource utilization and performance. The results show that while AXI DMA offers flexibility and scalability for continuous data streaming, BRAM-based integration achieves substantially lower latency and requires fewer hardware resources. The BRAM-based approach achieved over an 11-fold speedup compared to AXI DMA in the tested arithmetic operations.

The contributions of this work are threefold: (1) a detailed implementation of both integration strategies on the same SoC platform, (2) quantitative resource and performance evaluation, and (3) design insights into the trade-offs between throughput-oriented and latency-oriented accelerator architectures.

For future work, several directions can be pursued. First, more complex processing elements could be integrated to evaluate how these strategies scale with increased computational demands. Second, benchmarking with application-level workloads (e.g., digital filters, matrix multiplications, or neural network layers) would provide a broader view of practical performance. Finally, extending the evaluation to include energy consumption and thermal behavior would strengthen the design guidelines for power-sensitive embedded systems.

8. Author Contribution Statement

This study was carried out entirely by a single author. The author contributed to all stages of the work, including developing the research idea, designing the architectures, implementing the systems, collecting and analyzing data, reviewing the relevant literature, and preparing and critically revising the manuscript. No other individuals or institutions contributed to the study.

9. Ethics Committee Approval and Conflict of Interest

“There is no need for an ethics committee approval in the prepared article”

“There is no conflict of interest with any person/institution in the prepared article”

10. Ethical Statement Regarding the Use of Artificial Intelligence

No artificial intelligence-based tools or applications were used in the preparation of this study. The entire content of the study was produced by the author in accordance with scientific research methods and academic ethical principles.

11. References

- [1] A. Rios-Navarro, R. Tapiador-Morales, A. Jimenez-Fernandez, M. Dominguez-Morales, C. Amaya, and A. Linares-Barranco, "Performance evaluation over HW/SW co-design SoC memory transfers for a CNN accelerator," *J. Signal Process. Syst.*, vol. 91, no. 9, pp. 999–1012, Sep. 2019.
- [2] G. Tatar, S. Bayar, and İ. Çiçek, "Real-time multi-learning deep neural network on an MPSoC-FPGA for intelligent vehicles: Harnessing hardware acceleration with pipeline," *IEEE Trans. Intell. Veh.*, vol. 9, no. 6, pp. 5021–5032, Jun. 2024.
- [3] Y. Hao and S. Quigley, "The implementation of a deep recurrent neural network language model on a Xilinx FPGA," in *Appl. Reconfigurable Comput.*, vol. 10216, pp. 67–78, 2017.
- [4] G. Tatar and S. Bayar, "Real-time multi-task ADAS implementation on reconfigurable heterogeneous MPSoC architecture," *IEEE Access*, vol. 11, pp. 80741–80760, 2023.
- [5] Y. Wang, Z. Li, and H. Liang, "Scatter-gather DMA performance analysis within an SoC FPGA platform," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 17, no. 2, pp. 1–20, Apr. 2024.
- [6] H. Cilasun, "FPGA-accelerated simulation of variable latency memory for hardware/software co-design," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 28, no. 1, pp. 1–26, Jan. 2023.
- [7] G. Tatar *et al.*, "Recent advances in machine learning based advanced driver assistance system applications," *Microprocess. Microsyst.*, vol. 110, p. 105101, 2024.
- [8] J. Johnson, "Using the AXI DMA in Vivado," *FPGA Developer*, Aug. 2014.
- [9] K. Guerrero-Morejón, A. K. Bhattacharjee, and D. Atienza, "Embedded streaming hardware accelerators interconnect architectures and latency evaluation," *Electron.*, vol. 14, no. 8, p. 1513, Apr. 2025.
- [10] D. A. Jiménez-González, J. M. Arnau, and A. González, "Programmable FPGA-based memory controller: Evaluation of resource utilization and performance trade-offs," in *Proc. Int. Conf. Field-Programmable Logic Appl. (FPL)*, Sep. 2021.
- [11] E. Canto-Navarro, M. López-García, J. Font-Suñer, *et al.*, "AXI hardware accelerator for McEliece on FPGA," *IEEE Trans. Dependable Secure Comput.*, vol. 22, no. 2, pp. 1–14, Feb. 2025.
- [12] J. Boudjadar, "Dynamic FPGA reconfiguration for scalable embedded convolutional neural networks," *Future Gener. Comput. Syst.*, vol. 157, pp. 45–61, 2025.
- [13] B. Kieu-Do-Nguyen, N. T. Binh, C. Pham-Quoc, H. P. Nghi, N.-T. Tran, T.-T. Hoang, and C.-K. Pham, "Compact and low-latency FPGA-based number theoretic transform architecture for CRYSTALS Kyber post-quantum cryptography scheme," *Information*, vol. 15, no. 7, Art. no. 400, 2024.
- [14] P. Arya and S. H. Mokashi, "FPGA-accelerated RISC-V ISA extensions for efficient neural network inference on edge devices," arXiv preprint, 2025.
- [15] N. Irtija, J. Plusquellic, E. E. Tsiropoulou, J. Goldberg, D. Lobser, and D. Stick, "Design and analysis of digital communication within an SoC-based control system for trapped-ion quantum computing," *IEEE Trans. Quantum Eng.*, vol. 4, pp. 1–24, Art. no. 5500124, 2023.
- [16] C.-T. Axinte, A. Stan, and V.-I. Manta, "Embedded streaming hardware accelerators interconnect architectures and latency evaluation," *Electron.*, vol. 14, no. 8, p. 1513, 2025.
- [17] D. Berrazueta-Mena and B. Navas, "AHA: Design and evaluation of compute-intensive hardware accelerators for AMD-Xilinx Zynq SoCs using HLS IP flow," *Computers*, vol. 14, no. 5, p. 189, 2025.