Universidad San Francisco de Quito USFQ

Colegio de Ciencias e Ingenierías

Dragon AI: A Scalable AXI-Compatible AI Engine IP for Real-Time Edge Inference

Gabriel Sebastian Oña Chuquimarca

Ingeniería Electrónica

Trabajo de fin de Carrera presentado como requisito para la obtención del título de

Ingeniero en Electrónica

Quito, 1 de mayo del 2025

Universidad San Francisco de Quito USFQ

Colegio de Ciencias e Ingenierías

HOJA DE CALIFICACIÓN
DE TRABAJO DE FIN DE CARRERA

Dragon AI: A Scalable AXI-Compatible AI Engine IP for Real-Time Edge Inference

Gabriel Sebastian Oña Chuquimarca

Nombre del profesor, Título académico **Alberto Sánchez, PhD**

Quito, 1 de mayo del 2025

© DERECHOS DE AUTOR

Por medio del presente documento certifico que he leído todas las Políticas y Manuales de la Universidad San Francisco de Quito USFQ, incluyendo la Política de Propiedad Intelectual USFQ, y estoy de acuerdo con su contenido, por lo que los derechos de propiedad intelectual del presente trabajo quedan sujetos a lo dispuesto en esas Políticas.

Asimismo, autorizo a la USFQ para que realice la digitalización y publicación de este trabajo en el repositorio virtual, de conformidad a lo dispuesto en la Ley Orgánica de Educación Superior del Ecuador.

Nombres y apellidos: Gabriel Sebastian Oña Chuquimarca

Código: 00320597

Cédula de identidad: 1753402419

Lugar y fecha: Quito, 1 de mayo del 2025

ACLARACIÓN PARA PUBLICACIÓN

Nota: El presente trabajo, en su totalidad o cualquiera de sus partes, no debe ser considerado como una publicación, incluso a pesar de estar disponible sin restricciones a través de un repositorio institucional. Esta declaración se alinea con las prácticas y recomendaciones presentadas por el Committee on Publication Ethics COPE descritas por Barbour et al. (2017) Discussion document on best practice for issues around theses publishing, disponible en http://bit.ly/COPETheses.

UNPUBLISHED DOCUMENT

Note: The following capstone project is available through Universidad San Francisco de Quito USFQ institutional repository. Nonetheless, this project – in whole or in part – should not be considered a publication. This statement follows the recommendations presented by the Committee on Publication Ethics COPE described by Barbour et al. (2017) Discussion document on best practice for issues around theses publishing available on http://bit.ly/COPETheses.

RESUMEN

Los aceleradores de Redes Neuronales Profundas (DNN) se han convertido en un enfoque popular para introducir heterogeneidad en los sistemas de cómputo modernos. Sus implementaciones en FPGAs o ASICs a menudo superan a las arquitecturas de propósito general, como las GPUs y CPUs, en términos de eficiencia energética y rendimiento. Esto es especialmente crítico para las aplicaciones en tiempo real en el borde, que operan bajo estrictas restricciones de energía, latencia y área. Sin embargo, a medida que los aceleradores de inteligencia artificial (IA) se vuelven más especializados, los diseños basados en ASIC con frecuencia sacrifican flexibilidad en parametrización y reconfiguración. Presentamos *Dragon AI*, un acelerador de IA totalmente escalable en dimensionalidad espacial y número de bits del tipo de datos, con soporte para enteros y compatibilidad con la interfaz AXI. Nuestra plataforma demostró exitosamente una utilización ligera de recursos y bajo consumo de potencia en una Xilinx Kria KR260 con una arquitectura Zynq UltraScale+ MPSoC. El diseño logró una utilización de LUT y un consumo dinámico de energía en la lógica programable (PL) tan bajos como 7.52% y 0.017 W para un arreglo INT8 de 8×8, y tan altos como 62.62% y 0.174 W para un arreglo INT8 de 256×256. Además, implementamos con éxito un acelerador con un arreglo de 256×256x8b, igualando el tamaño del arreglo del TPU v1 de Google. Estos resultados demuestran que los dispositivos de borde pueden escalar dinámicamente el tamaño del arreglo sistólico para optimizar los cálculos de capas de aprendizaje automático según parámetros configurables, e incluso alcanzar dimensiones de arreglo comparables a los aceleradores de centros de datos.

Palabras clave: Acelerador de IA, FPGA, arreglo sistólico, computación en el borde, arquitectura parametrizable.

ABSTRACT

Deep Neural Network (DNN) accelerators have become a popular approach for introducing heterogeneity in modern computing systems. Their implementations on FPGAs or ASICs often outperform general-purpose architectures such as GPUs and CPUs in terms of energy efficiency and performance. This is especially critical for real-time edge applications, which operate under strict power, latency, and area constraints. However, as AI accelerators become more specialized, ASIC-based designs frequently trade flexibility in parameterization and reconfiguration. We present *Dragon AI*, a fully scalable spatial and data-width AI engine accelerator IP with integer support and AXI interface compatibility. Our platform successfully demonstrated lightweight resource utilization and power consumption on a Xilinx Kria KR260 with a Zynq UltraScale+ MPSoC architecture. The design achieved LUT utilization and programmable logic (PL) dynamic power consumption as low as 7.52% and 0.017W for an INT8 8×8 array, and as high as 62.62% and 0.174W for an INT8 256×256 array. Additionally, we successfully implemented a 256×256x8b array accelerator engine matching the array size of Google's TPU v1. These results show that edge devices can dynamically scale systolic array sizes to optimize ML layer computations across configurable parameters —even achieving array dimensions comparable to data center-grade accelerators.

Key words: AI accelerator, FPGA, systolic array, edge computing, parameterizable architecture.

Agradecimiento

El autor desea expresar su agradecimiento a la Universidad San Francisco de Quito por el respaldo brindado en términos de hardware e infraestructura. De manera especial, reconoce al Prof. Alberto Sánchez por su valioso acompañamiento y orientación a lo largo del proceso. Extiende también su más sincera gratitud a sus padres, Lucía y Galo, por su constante apoyo y confianza incondicional; a su hermana Mishelle, por ser una guía fundamental en su vida; y a Daniela, su pareja, por motivarlo a explorar nuevas perspectivas del mundo más allá del ámbito académico. Finalmente, agradece profundamente al Prof. Paul Chow, Prof. James Hoe, Prof. Jonathan Kelly y Prof. Mark Horowitz por haberlo introducido, respectivamente, al diseño de hardware, la aceleración con FPGA, la visión por computador para la percepción robótica, y la importancia del co-diseño de software y hardware en la aceleración del aprendizaje automático.

Table of Contents

Table of Figures

Table Index

Dragon AI: A Scalable AXI-Compatible AI Engine IP for Real-Time Edge Inference

Gabriel Oña, and Alberto Sánchez

***Abstract*—Deep Neural Network (DNN) accelerators have become a popular approach for introducing heterogeneity in modern computing systems. Their implementations on FPGAs or ASICs often outperform general-purpose architectures such as GPUs and CPUs in terms of energy efficiency and performance. This is especially critical for real-time edge applications, which operate under strict power, latency, and area constraints. However, as AI accelerators become more specialized, ASIC-based designs frequently trade flexibility in parameterization and reconfiguration. We present *Dragon AI*, a fully scalable spatial and data-width AI engine accelerator IP with integer support and AXI interface compatibility. Our platform successfully demonstrated lightweight resource utilization and power consumption on a Xilinx Kria KR260 with a Zynq UltraScale+ MPSoC architecture. The design achieved LUT utilization and PL dynamic power consumption as low as *7.52%* and *0.017W* for an INT8 8×8 array, and as high as *62.62%* and *0.174W* for an INT8 256×256 array. Additionally, we successfully implemented a 256×256×8b array accelerator engine matching the array size of Google's TPU v1. These results show that edge devices can dynamically scale systolic array sizes to optimize ML layer computations across configurable parameters—even achieving array dimensions comparable to data center-grade accelerators.**

***Index Terms*—AI accelerator, FPGA, edge computing, parameterizable architecture.**

I. Introduction

Heterogeneous computing has become increasingly important for a wide range of edge applications, driven by the rapid growth of compute-intensive artificial intelligence (AI) demands. Modern applications such as Automated Speech Recognition (ASR) [1], autonomous robotic perception and navigation [2], and the Internet of Things (IoT) [3] require efficient, high-performance computing on devices with limited energy resources.

Edge AI, in particular, is a paradigm where AI model training or inference happens directly on end devices [4]. Running models locally reduces bandwidth usage, improves latency, enhances data security, and increases energy efficiency [5], [6]. This approach avoids sending data to energy-intensive cloud resources, making it a suitable solution for real-time edge computing scenarios [7].

In this context, hardware acceleration for domain-specific AI tasks has recently gained significant attention. In particular, DNN accelerators [8]–[11] have proven to be efficient tools for both edge and cloud-based machine learning (ML) processing. However, many implementations fix in hardware layer-dependent variables such as systolic array size, dataflow architecture, and supported data types. To address this limitation, scalable and reconfigurable accelerators offer a flexible way to meet diverse needs.

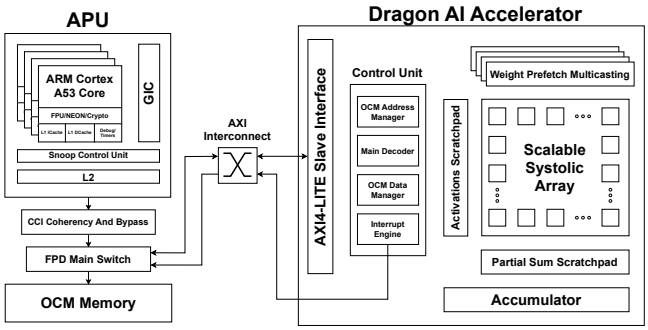

Fig. 1: Overview of the Dragon AI architecture.

In this work, we introduce *Dragon AI* [1], a fully scalable spatial and data-width AI accelerator IP package designed for flexible integration and parametrization within any compute pipeline. The AI engine incorporates a custom Instruction Set Architecture (ISA), facilitating seamless interaction between the software stack and the underlying hardware. Furthermore, it features an AXI4-Lite interface that manages control and data movement between the processing system (PS) and the programmable logic (PL).

Dragon AI supports integer multiply-accumulate (MAC) operations with configurable bitwidth and AXI-compatible interrupt controller. All implementation tests were conducted on a Xilinx Kria KR260 development board featuring the Zynq UltraScale+ MPSoC architecture. Our design demonstrates lightweight LUT utilization and reduced PL dynamic power consumption, with as little as *7.52%* and *0.017W* for an INT8 8×8 array, and as high as *62.62%* and *0.174W* for an INT8 256×256 array. These results show that edge devices can dynamically scale systolic array sizes to optimize ML layer computations—even achieving data-center-scale arrays comparable to Google's TPU v1 [12].

In summary, this work makes the following contributions:

- We designed and implemented a modular, scalable microarchitecture that allows users to configure systolic array dimensions and data type bitwidth through an interactive IP package.
- We defined a custom 64-bit ISA to facilitate efficient configuration and data movement between the PS and PL via an AXI4-Lite memory-mapped register interface.
- We incorporated an AXI-configurable interrupt controller to support efficient and parallelizable accelerator management.

Gabriel Oña is with the Department of Electrical Engineering, Universidad San Francisco de Quito, Quito, Ecuador (e-mail: gona@estud.usfq.edu.ec). Alberto Sánchez is with the same department (e-mail: asanchez@usfq.edu.ec).

Manuscript received May 2, 2025; revised May 2, 2025.

II. Background and Motivation

Local processing is especially important for real-time ML applications, where low system latency and high throughput are critical in time-sensitive scenarios. Selecting an appropriate compute platform is essential to meet the performance requirements of real-time edge AI applications [13].

In this section, we discuss key considerations for selecting DNN accelerators and the dependencies imposed by ML architectures.

A. DNN Accelerators

Several studies [2], [8]–[11] have demonstrated that specialized DNN accelerator architectures—such as systolic arrays and vector processors—consistently outperform general-purpose GPUs and CPUs for AI workloads. These architectures are particularly well-suited for edge AI applications, where power efficiency and performance are critical. Among available compute platforms, FPGAs and ASICs have shown the highest energy efficiency [2].

However, ASIC implementations require array dimensions and dataflow patterns to be fixed early in the hardware design process. Additionally, enabling flexibility in ASICs introduces area overhead due to the extra logic required for programmability. In contrast, FPGAs offer inherent reconfigurability, making them particularly advantageous for edge deployments where model characteristics and workload requirements can vary significantly.

B. Parametrizable Accelerators

Parametrization is a key factor in the development of DNN accelerators intended for highly heterogeneous computing environments. For example, Quickloop [14] is a framework that wraps the Chipyard [15] generator and can produce the RTL files needed to build a Gemmini [8] accelerator with an initial set of configuration parameters. Another example is Bifrost [16], a framework that enables end-to-end reconfigurability, facilitating the evaluation and optimization of DNN accelerator parameters. It generates MAERI [9] accelerators to improve performance metrics such as inference latency.

In this work, we propose the development of a framework that seamlessly integrates several parametrizable features from existing DNN accelerator frameworks and includes AXI compatibility to enable a straightforward integration and control of the generated accelerator by the PS on Xilinx SoC devices.

C. ML Layer Dependency

Similar to selecting an appropriate DNN accelerator, it is crucial to consider the target ML architecture that the platform will execute. Among the various types of DNNs, each benefits differently depending on the chosen dataflow and the degree of parallel processing supported by the array. Since accelerator platforms are often designed for reuse across varying matrix dimensions, it is advantageous to choose an array size that minimizes the need for matrix tiling.

For example, Convolutional Neural Networks (CNNs) often benefit from a weight-stationary (WS) dataflow and smaller systolic array sizes that align well with filter dimensions [13]. In contrast, Transformer architectures typically perform better with larger systolic arrays and an output-stationary (OS) dataflow, which are more suited to matrix-intensive operations such as attention mechanisms and feed-forward layers [17].

Therefore, building a parameterizable DNN accelerator is desirable to efficiently adapt to the varying demands of different ML architectures.

III. Dragon AI Methodology

A. Dragon AI Instruction Set Architecture Design

Designing an ISA for an AI engine accelerator requires a comprehensive analysis of all system operations and their interaction with the host processor. In general, these operations can be categorized into four groups: distribution, compute, accumulation, and collection [9].

The **distribution** phase manages data transfers across the memory hierarchy of the PS and the accelerator. Depending on the operation, the incoming data is stored as activations, weights, or partial sums.

The **compute** phase orchestrates the movement of activation data into the systolic array. It then retrieves the resulting partial sums into designated addresses within the partial sum scratchpad.

The **accumulation** phase performs arithmetic operations on partial sums. This is crucial when matrix dimensions exceed the systolic array size, requiring a tiling algorithm to partition matrices and accumulate partial results to form the final output.

The **collection** phase handles the transfer of computed partial sum results back to the PS.

Based on this operational analysis, a 4-bit opcode format was adopted, supporting up to 7 current instructions with the flexibility to scale up to 16. The following list provides a detailed description of each instruction:

- **IDLE (IDLE):** Puts the accelerator into a low-power sleep state to reduce energy consumption.
- **Weight Store (WGS):** Loads weight data row-wise into the systolic array. This instruction uses multicast controllers to simultaneously prefetch weights to all processing elements (PEs) within a row, identified by a unique ID.
- **Activation Store (ACS):** Transfers activation data from the processor to an specific address in the activation scratchpad.
- **Partial Sum Store (PSS):** Stores previously computed partial sum data at an specific address in the partial sum scratchpad.
- **Matrix Multiply Execute (MME):** Initiates matrix multiplication by passing activation data from its scratchpad to the systolic array input bus. The instruction increments the starting activation address until the end address is reached. Results are stored at an specific address in the partial sum scratchpad.
- **Partial Sum Accumulate (PSA):** Reads two values from the partial sum scratchpad and partitions them based on the data type bitwidth. The system performs accumulation

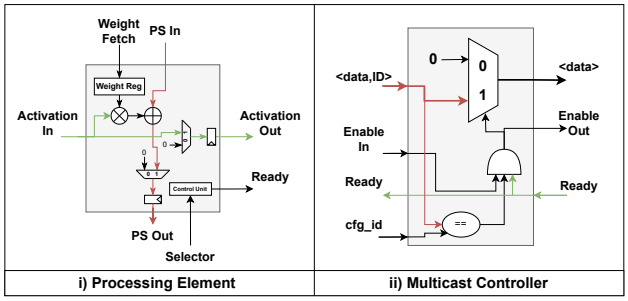

Fig. 2: Overview of the *i)* Processing Element (PE) and *ii)* Multicast Controller.

and writes the result back to the first input address location in the scratchpad.

- **Partial Sum Collect (PSC):** Retrieves data from an specific address in the partial sum scratchpad and returns it to the PS.

The complete *Dragon AI* ISA bitfields are included in *Appendix A*.

B. Microarchitecture Implementation

Following the design of the proposed ISA, we developed the block-level architecture of the entire system. This section outlines the key modules and microarchitectural decisions involved in integrating our ISA into a cohesive control unit and datapath.

1) Processing Elements (PEs): A processing element (PE) is the basic unit that performs MAC operations. We designed a PE using a WS dataflow, where the weight value is prefetched into an internal register. Each PE receives previously computed partial sums and activations from the neighboring PEs above and to the left, respectively. After performing a MAC operation, it sends the new partial sum downward and the used activation to the right. Figure 2.i shows the RTL structure of the PE module.

A two-bit state selector, along with supporting control logic, manages the PE's readiness and operational mode. Each PE operates in three states: *IDLE* (outputs are zeroed), *Weight Prefetch* (outputs remain zero while the weight register is enabled), and *Multiplication* (outputs are multiplexed between the activation and partial sum buses).

2) Multicast Controllers: A multicast controller is a module commonly used in Network-on-Chip (NoC) interfaces. It acts as a switch between a shared bus and multiple elements that require simultaneous access. In our implementation, each row of PEs—identified by a unique ID—coordinates access to the control unit's weight bus through a multicast controller. Figure 2.ii illustrates the RTL structure of the module.

This design enables batch prefetching, improving data-level parallelism [18]. Each multicast controller returns a ready signal to the control unit, computed as the reduction AND of the individual ready signals from all PEs in the row. Figure 3 shows the interface between multicast controllers and the systolic array.

3) Systolic Array: A systolic array is a 2D network of PEs that performs arithmetic operations in a cascaded manner. It

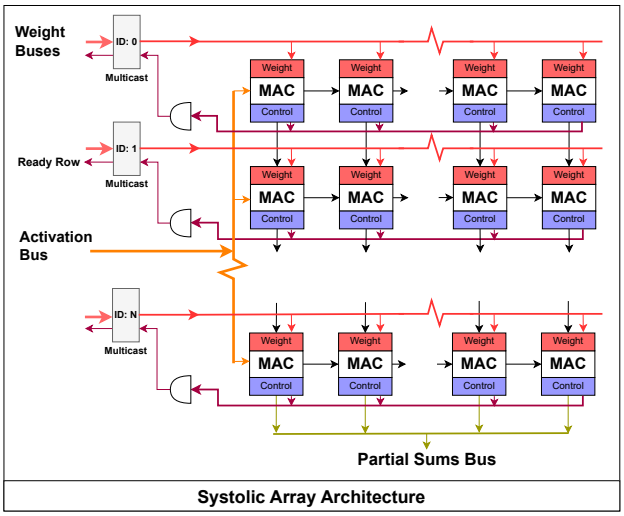

Fig. 3: Overview of the Systolic Array Architecture.

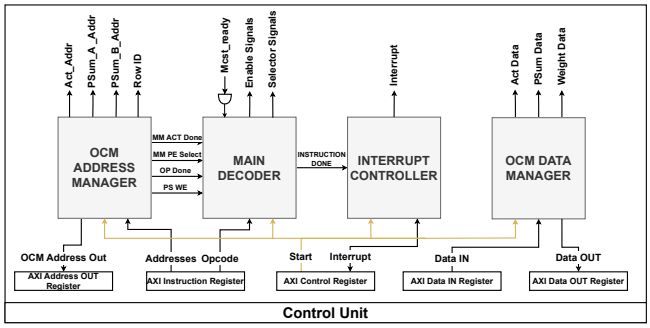

Fig. 4: Overview of the Dragon AI Control Unit.

is often regarded as the core of an AI engine accelerator. Specialized dataflows, array dimensions, and data types present different performance trade-offs depending on the architecture and workload of the ML model [8], [9]. Therefore, selecting appropriate systolic array parameters is a critical phase in accelerator design.

Our implementation adopts a WS dataflow with a multi-cycle instruction design. Figure 3 illustrates the RTL structure of the systolic array. Activations enter through the leftmost PEs and propagate horizontally, while each PE computes partial sums and passes them downward. The bottom row of PEs sends the final results to the partial sum scratchpad. The cascaded interfaces between PEs create a fully pipelined system, making it particularly efficient for matrix multiplication.

4) Scratchpads: Memory scratchpads are a type of on-chip memory characterized by low-latency data access. In our implementation, they store activation and partial sums in memory banks with a depth of 4096 words. The word size is determined by the product of the data type bitwidth and the number of border PEs.

For the activation scratchpad, we generated a template for single-port BRAM, which supports combinational read access on FPGAs. In contrast, for the partial sum scratchpad, two words must be retrieved simultaneously during accumulation. Therefore, we employed true dual-port BRAM, introducing a one-cycle latency for both read and write operations.

5) Control Unit: To maintain regularity and modularity in the design, all operations managed by the Control Unit were carefully classified into four independent modules. Figure 4 illustrates the RTL structure of the Control Unit. The following list outlines each proposed subdivision along with its specific responsibilities:

- **OCM Address Manager**—This module coordinates all addresses used across the different scratchpad memories. For distribution and collection instructions, it targets the addresses where data will be stored or retrieved. For compute and accumulation operations, it handles address movement for both the activations and partial sums scratchpads. Finally, it generates control signals to indicate the completion of operations to the Main Decoder.
- **Main Decoder**—This module decodes the incoming instruction and manages control signals within the AI engine. It synchronizes read and write enable signals based on the status of other modules in the datapath. Additionally, it generates an end-of-instruction signal to notify the Interrupt Controller when an operation has completed successfully.
- **OCM Data Manager**—This module manages data input and output transactions between the PS and the accelerator. Additionally, it can be flexibly replaced with a direct memory access (DMA) controller to route data directly to the AI engine.
- **Interrupt Controller**—This module manages end-of-instruction interrupt requests from the accelerator to the PS. It can be configured through an AXI control register that handles interrupt request (IRQ) enable and clear operations.

6) AXI4-Lite Core: The design integrates an AXI4-Lite interface to facilitate communication between the PS and the accelerator core. Five AXI memory-mapped registers were defined to coordinate operations, data movement, and interrupts of the AI engine. Since AXI4-Lite supports only 32-bit memory-mapped registers, each 64-bit field was split across two consecutive 32-bit registers. Table IV in *Appendix A* illustrates the bitfield structure of the AXI memory-mapped registers.

C. IP Package Parametrization

To create a custom IP package, we wrote a wrapper around the entire AI core and connected the memory-mapped registers to the AXI interface. We then linked these registers to the inputs of the top module and set global parameters for the width and height of the systolic array, as well as the data type bitwidth, which are displayed in the IP configuration visualizer. Figures 5, 6 show the top-level block diagram and IP package block desing of the Dragon AI accelerator.

IV. Implementation

In this section, we present several implementation scenarios of the proposed *Dragon AI* accelerator. We compare device utilization and power consumption under different parameters on a Xilinx Kria KR260 development board featuring a Zynq UltraScale+ MPSoC architecture.

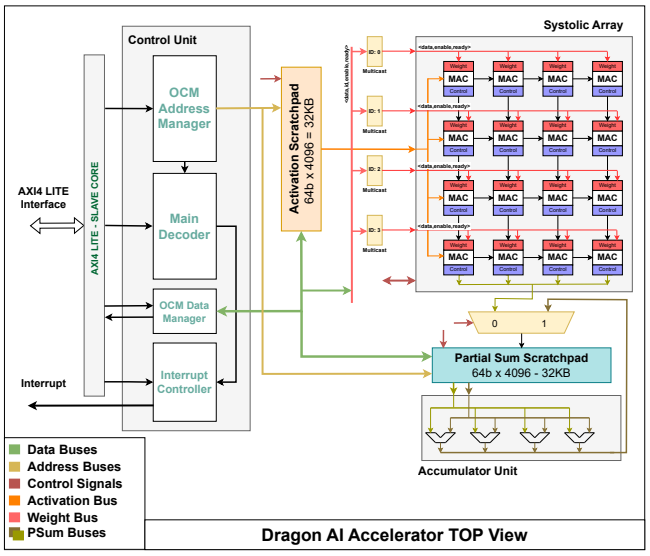

Fig. 5: Overview of the top-level block diagram of the Dragon AI accelerator

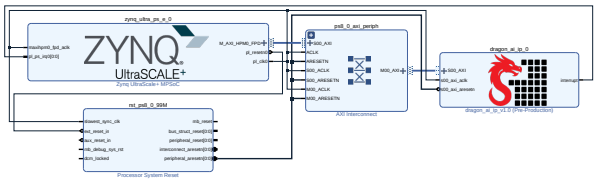

Fig. 6: Dragon AI IP package block desing

A. FPGA Resource Utilization

Table I summarizes the resource utilization across different systolic array sizes and data type bitwidths. The combination of these two parameters proved to be a non-trivial design choice for achieving efficient resource utilization.

- **DSP mapping**—For INT8 data type, the native Xilinx synthesis flow avoids mapping the RTL design to DSP slices in the Zynq fabric. This behavior is due to the fact that DSP slices are optimized for arithmetic operations, supporting dual 24-bit or quad 12-bit add/subtract/accumulate functions [19]. Consequently, all INT8 implementations resulted in zero DSP utilization, with a corresponding increase in LUT and FF usage.
- **Resource Constraints for Large Arrays**—We found that using INT32 data type with a 128×128 systolic array was not synthesizable. The design exceeded available BRAM resources, using 911 out of 288 available BRAM blocks. Therefore, for these dimensions, higher-capacity platforms such as the Virtex-7 VC709 or Versal VCK190 are preferred. As a result, this configuration is intended for deployment on cloud infrastructure rather than on edge devices.
- **Data-Center-Scale Systolic Array on the Edge**—We successfully deployed a 256×256 array supporting INT8 MAC operations. This array size is comparable to the systolic array implementation in Google's TPU v1, used for AI processing in data centers [12]. While newer generations of TPU also support floating-point operations

of various bitwidths, this result is a significant discovery, demonstrating that a data-center-grade array size can be deployed on an edge device.

Figure 7 illustrates device utilization implemented results.

Zynq Ultrascale+ MPSoC Utilization												
Data Type	INT 8				INT 16				INT 32			
Systolic Array Size	8x8	32x32	128x128	256x256	4x4	8x8	32x32	128x128	4x4	8x8	32x32	128x128
LUT [%]	7.52	9.69	19.49	62.62	5.32	5.98	8.76	28.20	5.81	6.48	11.71	—
LUTRAM [%]	7.26	7.26	7.26	7.26	7.26	7.26	7.26	7.26	7.26	7.26	7.26	—
FF [%]	1.18	2.28	7.28	35.94	0.64	0.68	1.13	11.58	0.70	0.82	1.73	—
BRAM [%]	5.21	5.56	5.56	5.56	5.21	5.56	5.56	5.56	5.56	5.56	5.56	—
DSP [%]	0.00	0.00	0.00	0.00	1.28	3.21	12.82	4.73	2.08	4.01	7.61	—
BUFG [%]	0.57	0.57	0.57	0.57	0.28	0.28	0.57	0.57	0.28	0.57	0.57	—
Total Utilization [%]	3.62	4.23	6.69	18.66	3.33	3.83	6.02	9.65	3.62	4.12	5.74	—

TABLE I: Kria KR260 Zynq Ultrascale+ MPSoC Utilization

B. FPGA Power Measurement

Table II shows the power consumption of the Zynq UltraScale+ MPSoC for different systolic array sizes and data type bitwidths. The results include the dynamic and static power consumption from the PL and PS regions. They highlight that selecting the best parameters to reduce power consumption on the target device is still a challenging task.

- **PS vs PL Dynamic Power Consumption**—We demonstrated that the dynamic PL power consumption is approximately 100× lower than the dynamic power consumption of the PS region. This result highlights the importance of power reduction techniques such as clock gating, power gating, and sleep modes for processors.
- **Power Benefits from Specialized Architectures**—As Moore's Law shows a decreasing trend due to the exponential growth of AI workloads and limited improvements in power efficiency from newer technology nodes, specialized architectures have become increasingly important. *Dragon AI* has demonstrated flexibility as a platform for conducting power consumption studies across various workloads, ML architectures, model quantization levels, and dataflows.

Zynq Ultrascale+ MPSoC Power Consumption												
Data Type	INT 8				INT 16				INT 32			
Systolic Array Size	8x8	32x32	128x128	256x256	4x4	8x8	32x32	128x128	4x4	8x8	32x32	128x128
Dynamic without PS [W]	0.017	0.021	0.050	0.174	0.014	0.017	0.027	0.078	0.016	0.021	0.032	—
Dynamic PS [W]	2.415	2.415	2.415	2.415	2.415	2.415	2.415	2.415	2.415	2.415	2.415	—
Static PL [W]	0.301	0.301	0.301	0.301	0.301	0.301	0.301	0.301	0.301	0.301	0.301	—
Total [W]	2.733	2.737	2.766	2.890	2.730	2.733	2.743	2.794	2.732	2.737	2.748	—

TABLE II: Kria KR260 Zynq Ultrascale+ MPSoC Power Consumption

V. Conclusion

We presented *Dragon AI*, a flexible AI engine with parameterizable systolic array dimensions and integer MAC operations bitwidth. It features a custom ISA with a WS dataflow and AXI support. We demonstrated that this platform can be used for various studies on device resource utilization and power consumption across different parameter selections. We successfully deployed a 256×256 array supporting INT8 MAC operations, showing that a data-center-grade array size—comparable to Google's TPU v1—can be implemented at the edge.

Acknowledgment

The first author would like to thank Universidad San Francisco de Quito for providing hardware and infrastructure support. Special thanks to Prof. Alberto Sanchez for his invaluable support and guidance. The author also expresses deep gratitude to Prof. Paul Chow, Prof. James Hoe, Prof. Jonathan Kelly, and Prof. Mark Horowitz for introducing him to hardware design, FPGA acceleration, computer vision for robotic perception, and the importance of software/hardware co-design in ML acceleration, respectively.

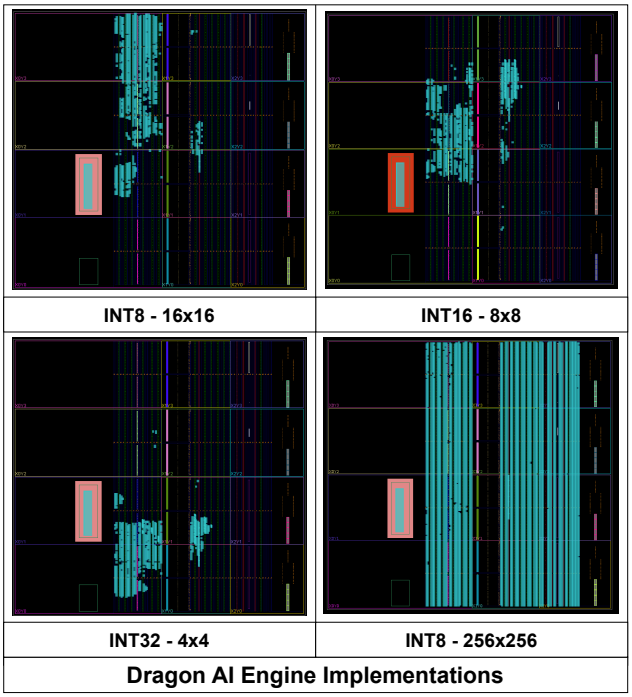

Fig. 7: Zynq Ultrascale+ MPSoC Utilization.

References

[1] D. M. Chan, S. Ghosh, D. Chakrabarty, and B. Hoffmeister, "Multi-modal pre-training for automated speech recognition," in *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022, pp. 246–250.

[2] A. Suleiman, Z. Zhang, L. Carlone, S. Karaman, and V. Sze, "Navion: A 2-mw fully integrated real-time visual-inertial odometry accelerator for autonomous navigation of nano drones," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 4, pp. 1106–1119, 2019.

[3] A. Rocha, M. Monteiro, C. Mattos, M. Dias, J. Soares, R. Magalhães, and J. Macedo, "Edge ai for internet of medical things: A literature review," *Computers and Electrical Engineering*, vol. 116, p. 109202, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790624001307

[4] S. S. Gill, M. Golec, J. Hu, M. Xu, J. Du, H. Wu, G. K. Walia, S. S. Murugesan, B. Ali, M. Kumar, K. Ye, P. Verma, S. Kumar, F. Cuadrado, and S. Uhlig, "Edge ai: A taxonomy, systematic review and future directions," *Cluster Computing*, vol. 28, no. 1, Oct. 2024. [Online]. Available: http://dx.doi.org/10.1007/s10586-024-04686-y

[5] X. Wang, Z. Tang, J. Guo, T. Meng, C. Wang, T. Wang, and W. Jia, "Empowering edge intelligence: A comprehensive survey on on-device ai models," *ACM Comput. Surv.*, vol. 57, no. 9, Apr. 2025. [Online]. Available: https://doi.org/10.1145/3724420

[6] S. Dhar, J. Guo, J. J. Liu, S. Tripathi, U. Kurup, and M. Shah, "A survey of on-device machine learning: An algorithms and learning theory perspective," *ACM Trans. Internet Things*, vol. 2, no. 3, Jul. 2021. [Online]. Available: https://doi.org/10.1145/3450494

[7] C. R. P, A. S. Ahmed, V. Divya, Y. Nagendar, A. Mohan, and A. Atheeswaran, "Real-time signal processing in iot-based embedded systems using hybrid ai-enhanced edge computing," in *2024 International Conference on Artificial Intelligence and Quantum Computation-Based Sensor Application (ICAIQSA)*, 2024, pp. 1–7.

[8] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao *et al.*, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *2021 58th*

ACM/IEEE Design Automation Conference (DAC). IEEE, 2021, pp. 769–774.

[9] H. Kwon, A. Samajdar, and T. Krishna, "Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects," *ACM Sigplan Notices*, vol. 53, no. 2, pp. 461–475, 2018.

[10] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[11] A. Carsello, K. Feng, T. Kong, K. Koul, Q. Liu, J. Melchert, G. Nyengele, M. Strange, K. Zhang, A. Nayak, J. Setter, J. Thomas, K. Sreedhar, P.-H. Chen, N. Bhagdikar, Z. Myers, B. D'Agostino, P. Joshi, S. Richardson, R. Bahr, C. Torng, M. Horowitz, and P. Raina, "Amber: A 367 gops, 538 gops/w 16nm soc with a coarse-grained reconfigurable array for flexible acceleration of dense linear algebra," in *2022 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, 2022, pp. 70–71.

[12] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," *SIGARCH Comput. Archit. News*, vol. 45, no. 2, p. 1–12, Jun. 2017. [Online]. Available: https://doi.org/10.1145/3140659.3080246

[13] V. Sze, Y. Chen, T. Yang, and J. Emer, *Efficient Processing of Deep Neural Networks*, ser. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2020. [Online]. Available: https://books.google.com.ec/books?id=wLHuDwAAQBAJ

[14] T. Mahmood, K. Inayat, and J. Chung, "Quickloop: An efficient, fpga-accelerated exploration of parameterized dnn accelerators," in *2023 32nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2023, pp. 327–328.

[15] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton, P. Rigge, C. Schmidt, J. Wright, J. Zhao, Y. S. Shao, K. Asanović, and B. Nikolić, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.

[16] A. Stjerngren, P. Gibson, and J. Cano, "Bifrost: End-to-end evaluation and optimization of reconfigurable dnn accelerators," in *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2022, pp. 288–299.

[17] Q. Liu, M. Zapater, and D. Atienza, "Matrixflow: System-accelerator co-design for high-performance transformer applications," 2025. [Online]. Available: https://arxiv.org/abs/2503.05290

[18] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, ser. The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier Science, 2017. [Online]. Available: https://books.google.com.ec/books?id=MBQFuAEACAAJ

[19] Xilinx Inc., *7 Series DSP48E1 Slice User Guide (UG479)*, AMD Xilinx, 2023, version 1.13, November 20, 2023. [Online]. Available: https://docs.xilinx.com/r/en-US/ug479_7Series_DSP48E1

APPENDIX A
DRAGON AI INSTRUCTION SET ARCHITECTURE

Dragon AI ISA								
Data Processing	**63:60 Opcode**				**59:0 X**			
IDLE (IDLE)	0000				X			
Memory Type 1	**63:60 Opcode**	**59:48 X**	**47:36 X**	**35:32 ROW ID**		**31:0 Address BRAM**		
Weight Store (WGS)	0001	X	X	ID		Data Address		
Memory Type 2	**63:60 Opcode**	**59:48 X**	**47:44 X**	**43:32 Destination In-memory**		**31:0 Address BRAM**		
Activation Store (ACS)	0010	X	X	Act Scratch Addr		Data Address		
Partial Sum Store (PSS)	0011	X	X	Psum Scratch Addr		Data Address		
Compute Instructions	**63:60 Opcode**	**59:48 X**	**47:44 X**	**43:32 Address End Act**	**31:28 X**	**27:16 Address Start Act**	**15:12 X**	**11:0 Address PS**
Matrix Multiply (MME)	0100	X	X	Address End Act	X	Address Start Act	X	Psum Start Addr
Accumulate Instructions	**63:60 Opcode**	**59:48 X**		**47:32 X**	**31:28 X**	**27:16 Address Psum 1**	**15:12 X**	**11:0 Address Psum 2**
PS Accumulate (PSA)	0101	X		X	X	Address Psum 1	X	Address Psum 2
Collect Instructions	**63:60 Opcode**	**59:48 X**	**47:44 X**	**43:32 Address Psum**		**31:0 Address BRAM**		
PS Collect (PSC)	0110	X	X	Address Psum		Data Address		

TABLE III: Dragon AI Instruction Set Architecture Bitfields

AXI Memory-Mapped Registers				
Instruction [63:0]				
Registers	Reg_1	Reg_0		
Fields	Instruction [63:32]	Instruction [31:0]		
Control [31:0]				
Registers	Reg_7			
Fields	X	Control[2] - IRQ_Clear	Control[1] - IRQ_Enable	Control[0] - Start
Data IN [63:0]				
Registers	Reg_3	Reg_2		
Fields	Data IN [63:32]	Data IN [31:0]		
Address OUT [31:0]				
Registers	Reg_4			
Fields	Address OUT [31:0]			
Data OUT [63:0]				
Registers	Reg_6	Reg_5		
Fields	Data IN [63:32]	Data IN [31:0]		

TABLE IV: AXI Memory-Mapped Registers