Received 11 April 2023, accepted 10 May 2023, date of publication 15 May 2023, date of current version 24 May 2023.

*Digital Object Identifier 10.1109/ACCESS.2023.3276411*

RESEARCH ARTICLE

# The Design of Optimized RISC Processor for Edge Artificial Intelligence Based on Custom Instruction Set Extension

**HYUN WOO OH AND SEUNG EUN LEE, (Senior Member, IEEE)**
Department of Electronic Engineering, Seoul National University of Science and Technology, Seoul 01811, South Korea

Corresponding author: Seung Eun Lee (seung.lee@seoultech.ac.kr)

This work was supported by the Ministry of Science and ICT (MSIT), South Korea, through the Information Technology Research Center (ITRC) Support Program, supervised by the Institute for Information and Communications Technology Planning and Evaluation (IITP), under Grant IITP-2023-RS-2022-00156295.

**ABSTRACT** Edge computing is becoming increasingly popular in artificial intelligence (AI) application development due to the benefits of local execution. One widely used approach to overcome hardware limitations in edge computing is heterogeneous computing, which combines a general-purpose processor with a domain-specific AI processor. However, this approach can be inefficient due to the communication overhead resulting from the complex communication protocol. To avoid communication overhead, the concept of an application-specific instruction set processor based on customizable instruction set architecture (ISA) has emerged. By integrating the AI processor into the processor core, on-chip communication replaces the complex communication protocol. Further, custom instruction set extension (ISE) reduces the number of instructions needed to execute AI applications. In this paper, we propose a uniprocessor system architecture for lightweight AI systems. First, we define the custom ISE to integrate the AI processor and GPP into a single processor, minimizing communication overhead. Next, we designed the processor based on the integrated core architecture, including the base core and the AI core, and implemented the processor on an FPGA. Finally, we evaluated the proposed architecture through simulation and implementation of the processor. The results show that the designed processor consumed 6.62% more lookup tables and 74% fewer flip-flops while achieving up to 193.88 times enhanced throughput performance and 52.75 times the energy efficiency compared to the previous system.

**INDEX TERMS** AI processor, application specific instruction processor, custom instruction set extension, edge computing, embedded systems, processor core, reduced instruction set computer.

## I. INTRODUCTION

Nowadays, ongoing advances in semiconductor process technology encourage a variety of digital systems to embed more dense circuits by reducing the resources, e.g., power consumption and area usage, while improving the time-related performance of basic elements for digital chip implementation. This trend emboldens recent attempts to adopt complex algorithms to a variety of applications. Artificial intelligence (AI) algorithms, e.g., machine learning (ML) and deep neural network (DNN), are complex algorithms that are being actively applied to these applications.

Applications based on lightweight embedded systems are one of the fields where AI algorithms are actively applied. Edge computing has become one of the major topics in the development of AI applications due to the benefits of replacing the cloud execution with the local execution, such as reduced network bandwidth usage, enhanced privacy protection, and minimized storage waste [1], [2]. Many ongoing studies introduce several methods for distributing the workloads of AI algorithms to lightweight systems [3], [4], [5]. One of the main challenges is overcoming the hardware

The associate editor coordinating the review of this manuscript and approving it for publication was Stavros Souravlas.

constraints of lightweight systems, such as low performance and area limitations.

Heterogeneous computing is one of the methods to overcome these limitations. The general-purpose processor (GPP) is essential to most digital systems, but executing most parts of the AI algorithms with only the single GPP is difficult because most GPPs are optimized to perform various types of simple operations sequentially, while AI algorithms are composed of numerous limited types of operations [6]. Therefore, not only the lightweight systems but also the cloud computing-based high-performance systems that target the AI application adopt heterogeneous computing with GPP and hardware unit to accelerate arithmetic operations through parallelization [7], [8].

The main difference in hardware acceleration between these systems is the diversity of executable AI algorithms. The high-performance systems include domain-general hardware units such as general-purpose graphics processing unit (GPGPU), and tensor processing unit (TPU), in general [7]. In contrast, most embedded systems usually include hardware units for specific AI algorithms because of the resource constraints involved in lightweight systems [9], [10]. As edge AI systems generally target domain-specific applications, trading off versatility for resource utilization, e.g., power consumption and area usage, is an effective strategy for the systems [11]. In fact, many studies have been conducted to design modular AI processors for heterogeneous computing-based edge AI applications [7], [8], [9].

Nevertheless, AI systems composed of modularized heterogeneous processors still involve drawbacks stem from certain characteristics. Communication protocols used for handshaking between devices, such as universal asynchronous receiver/transmitter (UART), serial peripheral interface (SPI), and peripheral component interconnect express (PCIe), are one of the major characteristics that generate the inefficiencies in the system. As the data rate of these protocols is much slower than the parallel communication inside the processors running on each operating frequency, significant overheads are required for handshaking between both processors.

The increased workload allocation for communication results in degraded system performance. Moreover, synchronizing the processors requires additional controller units for both for communication in compliance with the protocol. These controllers increase the area of each processor, resulting in higher energy consumption for the entire system. As these drawbacks, e.g., throughput degradation and electric energy dissipation, are critical in lightweight systems [12], minimizing the inefficiency generated by the communication protocol is necessary.

Application-specific instruction set processor (ASIP) based on customizable instruction set architecture (ISA) for GPP is one of the concepts suggested to avoid the inefficiencies caused by the communication overheads in modularized architecture. Fig. 1 shows the system architectures of both heterogeneous computing systems and uniprocessor systems based on ASIP. By integrating the AI core into the core of the GPP, the roles of either the AI processor and the GPP, which are executed separately on previous heterogeneous computing systems, can be executed by only a single processor. This characteristic reduces the communication overheads and eliminates the necessity for communication controllers as the protocol is replaced with the on-chip communication running synchronously on the operating frequency of the core.

The ASIP concept shares the view with the concept based on bus topology [13], [14] that minimizes the overheads by simplifying and boosting communication performance through chip-level integration. Despite this similarity, the ASIP concept has advantages in resource utilization derived from the ISE. Through custom ISE, the number of instructions to execute AI applications becomes lower because a number of memory load/store instructions, which deliver the orders to the AI accelerator linked to the system bus, is converted to a small number of custom instructions. This characteristic reduces the memory usage for storing instruction codes and increases the throughput of the system [15], [16]. This advantage made the ASIP concept a reasonable choice despite the difficulty of the design process originating from the core modification.

RISC-V and MIPS are suitable ISAs for realizing the ASIP concept. Both ISAs provide productivity in the custom ISE definition process through the guidelines in the ISA documentation [17], [18]. Indeed, several studies applied the ASIP concept with one of these ISAs for systems targeting AI applications [19], [20], [21], [22]. However, most of those studies concentrate on applying this concept to complex AI algorithms such as deep learning and convolutional neural networks (CNN) [23], [24], [25], [26], [27], [28]. These heavy algorithms consume huge resources, making them inappropriate for lightweight systems due to hardware limitations [29]. For this reason, choosing an appropriate algorithm that consumes affordable resources is necessary to apply the concept to lightweight systems.

Another thing to consider is an additional workload for rewriting the source codes of the application. Custom instructions are not created by compiling the source codes written in general syntaxes of compilers. Hence, software developers need to mutate the legacy source codes to the new inline assembly codes to operate the AI core. Defining the custom ISE similar to the ISA of previous AI processors is necessary because conserving the previous mechanism lowers the additional workloads for rewriting the source codes.

In this paper, we propose a uniprocessor system architecture for lightweight AI systems. To minimize the communication overhead between the AI processor and the general-purpose processor, we selected the AI processor suitable for lightweight embedded systems and defined the custom ISE to design the architecture that integrates the GPP and the AI processor into a single processor.

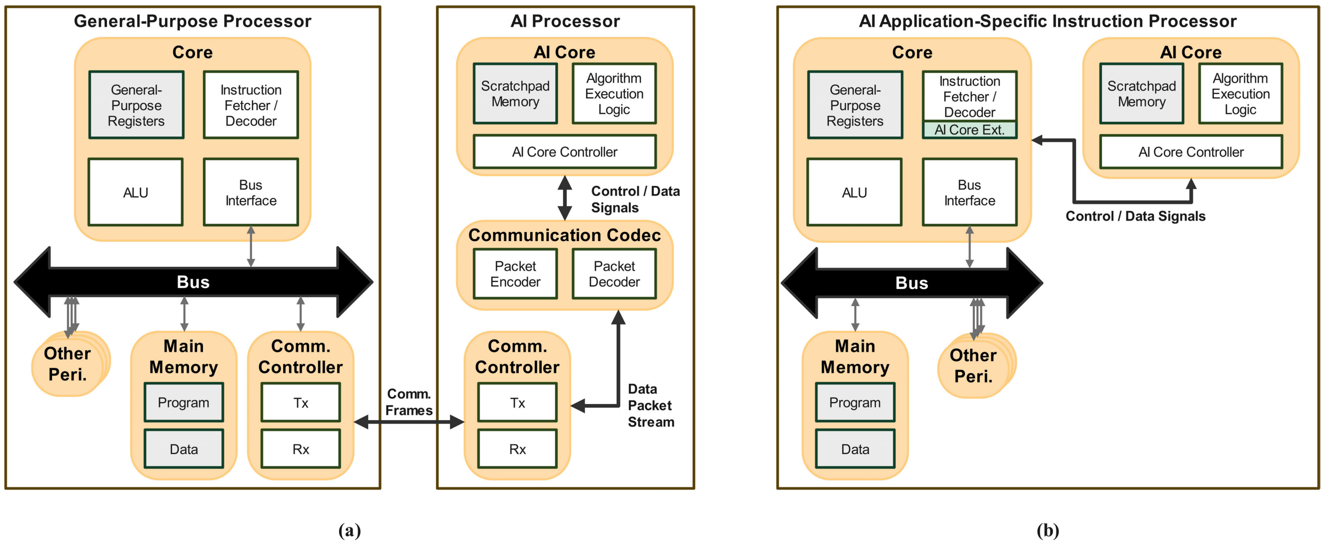

**FIGURE 1.** **(a) The heterogeneous system architecture. (b) The ASIP-based system architecture.**

The ISE is compatible with MIPS ISA, which includes reserved coprocessor definitions for the design-specific ISE. Further, we designed the integrated core architecture of the processor that executes the ISE. The core architecture includes the processor core and the AI core, which are operated by the synchronous clock.

According to this architecture, complex communication is replaced with simple data transferring through communication based on multiple internal wires and buffers in only a few clock periods. To verify the system and the core architecture, we designed the processor to a register-transfer level (RTL) written in Verilog HDL and build up the verification environment with a field-programmable gate array (FPGA) implementation.

Next, we developed the software library containing inline assembly codes that perform the same operation as the legacy library functions, which are developed for the previous modular AI processor. Finally, we evaluated the proposed architecture and the designed processor by executing sample applications that are already verified for the heterogeneous computing system with the previous AI processor.

The main contributions of this work are listed as follows:

- Customized ISE to minimize the overhead caused by the complex communication protocol in the modularized AI system.
- The AI coprocessor architecture and AI core architecture optimized for 32-bit MIPS core.
- The core architecture for GPP with integrated AI coprocessor compatible with customized ISE.
- The unified processor architecture designed to RTL.
- The software library to control the AI coprocessor.
- Evaluation of the processor architecture by realizing the designed processor to a field-programmable gate array (FPGA) implementation.

In order to describe our work clearly, the rest of the paper is categorized as follows. Section II reviews the related works that motivated our research. Section III explains the base architecture of the target modular AI processor, which is used to extract and optimize the AI core for the AI coprocessor. Section IV presents the custom ISE definitions and operations, which are based on the operation of the AI processor. Section V presents the integrated core architecture with the detailed operation of custom ISE, as well as unified processor architecture with integrated core architecture. Section VI describes the verification environment and evaluates the proposed architecture. Section VII concludes our research by summarizing our work and outlining the future endeavors to extend this work.

## II. RELATED WORKS

### A. FPGA-BASED LIGHTWEIGHT CNN ACCELERATORS

Many studies have been conducted to apply edge computing for AI, using FPGA-based heterogeneous computing methods. Kim et al. [30] proposed hardware acceleration for lightweight systems based on SqueezeNet [31], which is a lightweight CNN model for embedded systems. The authors transformed 32-bit floating-point arithmetic to 8-bit integer arithmetic to reduce the intensity of SqueezeNet and designed the parallelized architecture. Xia et al. [32] propose SparkNet, which utilizes depthwise separable convolution to reduce the intensity of the CNN. To optimize the CNN for the embedded systems, the authors of this work quantized the parameters to 16-bit integers and designed the accelerator with pipelined architecture, applying optimal multilevel parallelism. Despite the astonishing performance of the CNN-based heterogeneous accelerators proposed in these studies, the baseline intensity and numerous parameters of multi-layered neural networks still make challenging to adopt these works on ultra-lightweight embedded systems due to power consumption, energy dissipation, and area usage originating from calculation logics and external memory.

### B. LIGHTWEIGHT ML ACCELERATORS

As the CNNs are not suitable for resource-limited ultra-lightweight systems, lightweight ML accelerators and processors have been designed and introduced. General Vision introduces a neuron-inspired pattern recognition chip based on k-nearest neighbor (k-NN) algorithm, the CM1K, which is an application-specific integrated circuit (ASIC) [33]. Abeysekara et al., and Suri et al. [34], [35] developed heterogeneous systems for AI applications with lightweight GPP and the CM1K. These works overcome the computing performance limitations of the lightweight GPP but still have drawbacks related to energy dissipation and throughput degradation originating from the complex communication protocol.

To address these issues, Intel proposes a processor module named Curie, including pattern matching engine (PME) that accelerates the k-NN algorithm and are manipulated by bus transactions [36]. Although manipulating the accelerator through a bus topology increases the throughput performance and reduces the energy dissipation, this method still has memory access overheads originating from bus transactions.

On the other hand, several studies proposed utilizing burst access in bus topology to reduce memory access bottlenecks. Borelli et al. [37] proposed the k-NN accelerator for heterogeneous computing that uses the scratchpad memory through direct memory access (DMA) connected to an advanced extensible interface (AXI) stream port inside the processing system. Hussain et al. [38] designed the accelerator using a parallel first-in-first-out (FIFO) module to reduce memory access bottlenecks by burst transactions. In both architectures, data transactions between the GPP and the accelerator are processed through burst read and write operations, making the data transactions faster. Li et al. [39] represent the k-NN accelerator accessed by the main processor through the AXI slave interface and compose the scratchpad memory as dynamic random access memory (DRAM). In this study, the accelerator directly controls the DMA with burst operations to manipulate the DRAM controller through the AXI stream master port. These studies significantly reduced the memory access bottlenecks but are not optimized for ultra-lightweight embedded systems, as these designs exploit not only the on-chip memory but also the external memories, resulting in a heavier system in terms of area usage, power consumption, and energy dissipation and leading to throughput degradation due to long memory access latency.

Different from these works, we concentrate on these terms:

- Designing an optimized architecture to reduce bottlenecks of the memory access, which is required for AI computation, by eliminating structural overheads caused by bus transactions while avoiding the use of external memories, which make the system heavier, making the system inappropriate for ultra-lightweight systems.
- Optimizing the internal architecture of the k-NN calculation logic through parallelism directly reflecting the data rate and timing constraints of the general-purpose core inside the processor.
- Provide the reconfigurability that reflects the k-NN parameters to adopt the AI core by varying constraints of the lightweight systems.

## III. BACKGROUND: INTELLINO AI PROCESSOR

Before designing the processor based on the proposed system architecture for edge AI applications, we chose the appropriate AI processor, Intellino, which is designed for heterogeneous computing systems. Intellino is a reconfigurable AI processor based on the k-NN algorithm, which is a lightweight ML algorithm based on distance calculation and operates with the SPI, which is a common interface for high-speed communication in lightweight embedded systems [40], [41].

Fig. 2 shows the top architecture of the Intellino, which consists of the SPI slave controller, packet decoder and encoder, neuron controller, and classifier. As the Intellino is the slave device, which cannot operate independently, the SPI slave controller performs communication dependent on the generated clock, i.e., serial clock (SCK), from the host. After receiving the one-byte SPI frame, the controller sets the signal called rx_cplt and delivers the received data to the packet decoder. The packet decoder converts the received data to the instruction for the neuron controller by buffering the data and observing the specific byte sequences based on the protocol of the Intellino.

The instruction set for the neuron controller is defined as the operations based on register map definition. Thus, the single instruction for the neuron controller is represented as the control signals for the register file, such as register address (reg_addr), register write enable (reg_we), register write data (reg_wd), and register read enable (reg_re). The neuron controller generates the control signals and data signals for neuron cells and sends the calculation results of the k-NN algorithm generated by the classifier. The neuron cells and the classifier make up the k-NN calculator.

Fig. 3 shows the architecture of the k-NN calculation process. The k-NN algorithm is calculated by searching the category-distance pair, which has the shortest distance from the inference data vector, from the pre-trained dataset. The k-NN calculator of the Intellino has an architecture that performs the calculation efficiently. At first, the distance data of each dataset and current inference data vector are calculated by each neuron cell. Next, The classifier submodule calculates the category with the lowest distance by comparing the distance data using the lower distance selectors that are constructed to a multi-level inverted tree structure. Through this structure, the data pair with the lowest distance is derived by the selector in the last level.

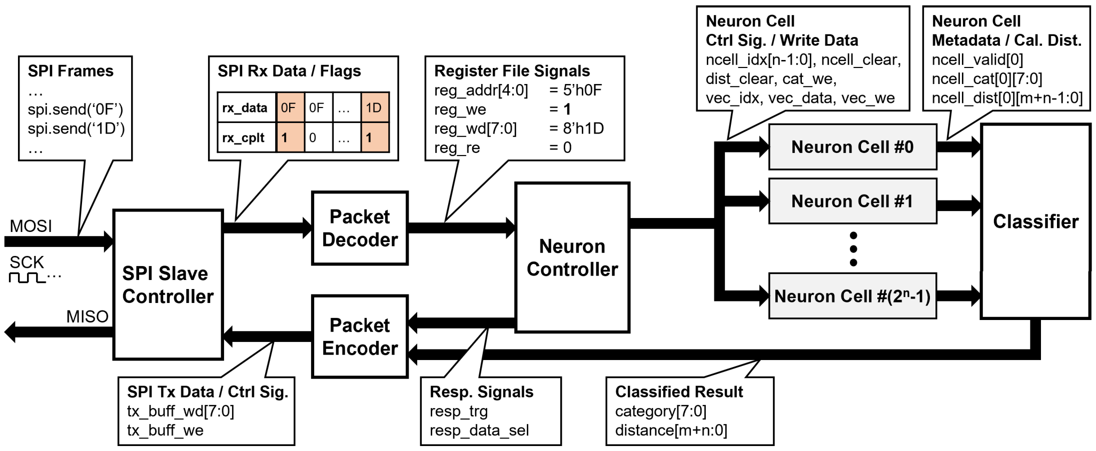

**FIGURE 2.** Top architecture of the Intellino.

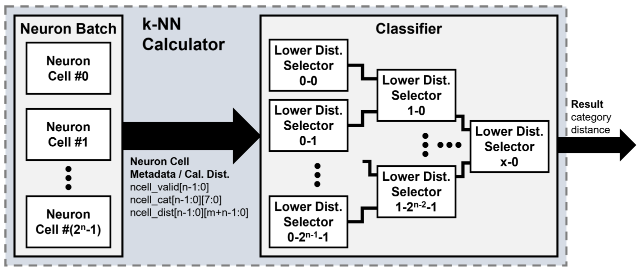

**FIGURE 3.** The architecture of the k-NN execution logic of the Intellino.

Fig. 4 shows the internal architecture of the k-NN submodules. The main role of the neuron cells is storing the pre-trained dataset while in the training process and calculating the distance between the stored vector and the input data stream while in the inference process. To store the dataset, each neuron cell has a scratchpad memory (SPM) for data vector, register for category data, and register for checking the validity of the stored data in SPM and category. These storages are sequentially written by one byte when the data in the neuron cell identifier (NID) register (ncell_idx) in the neuron controller matches the identifier of the current neuron cell and the appropriate write enable signal is set by the neuron controller. Removing the stored data is executed by clearing the flag set by the neuron controller. Distance calculation runs similarly to the dataset storing. The distance accumulator in each neuron cell calculates the distance between components in the stored vector and inference vector on the current component index and accumulates the calculated distance to the previous distance value. As the inference process proceeds through byte sequence writing, the total distance calculation occurs simultaneously with the reception of the last component of the inference vector. At the time, the classifier takes the data batches, which are composed of the valid flag, category, and calculated distance, from the neuron batch and passes the two batches to the selectors at the first level. The selector chooses the data batch from two batches. Basically, the selector chooses the batch containing a lower distance than others when both batches are valid. In the condition that one of the batches is not valid, the selector chooses the valid data batch. Through several steps, the shortest data pair is derived from the multiple batches.

According to the architecture, the accuracy performance of the k-NN algorithm depends on the count of the dataset and the length of the data vector. Intellino adopts reconfigurable architecture with two customized parameters that reflect the dependents: $2^n$ as the number of neuron cells and $2^m$ as the length of the data vector. This characteristic provides flexibility to compose the system with the Intellino for a variety of lightweight embedded systems. Nevertheless, the communication overheads are still generated by the SPI-based protocol as the speed of the k-NN calculator, which operates by parallel wires synchronous to the internal clock, is much higher than the speed of the protocol, which operates by serial communication synchronous to the slow external clock. Therefore, eliminating the overheads to improve throughput and minimize the area by embedding the k-NN calculator of the Intellino into the processor core is an appropriate strategy. To achieve this in our work, we designed an integrated core architecture that utilizes the Intellino as the coprocessor, which is operated by the custom ISE that we defined.

## IV. ISE FOR THE AI PROCESSOR

In MIPS ISA, the custom ISE is defined as custom coprocessor instructions. The MIPS processor can contain up to four coprocessors according to the MIPS ISA definition. Except for the coprocessor 0, which is an interrupt and exception controller, and the coprocessor 1, which is defined for floating-point extension, the operation of the coprocessor varies by the designer. Fig. 5 shows the schematized concept of the custom coprocessor. The coprocessor in MIPS ISA is the concept represented as the additional core that executes instruction codes on the custom ISE that shares the instruction fetch (IF) process and memory read/write (MEM) process. In common, the custom coprocessor embeds register files (CPnR) as shown in figure 5. Some coprocessor

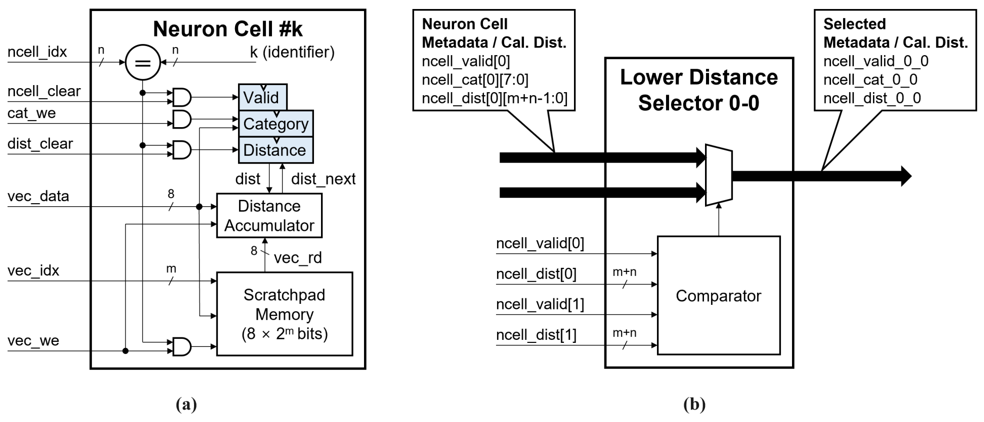

**FIGURE 4.** **(a)The internal architecture of the neuron cell. (b)The internal architecture of the lower distance selector.**

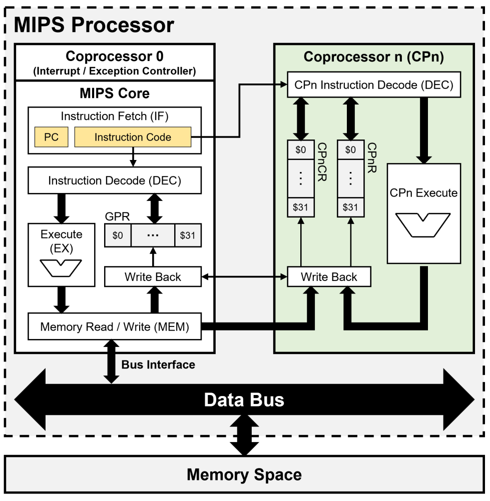

**FIGURE 5.** **The concept of the custom coprocessor in MIPS ISA.**

embeds the additional register files to control the coprocessor (CPnCR). These characteristics provide functional safety on the compiled binary originating from source codes written in a high-level language, which specifies the role of each register in general-purpose registers (GPR), by ensuring functional independency of both cores. To form data communication between the MIPS core and the coprocessor, the MIPS ISA offers guidelines for defining the sample instructions that perform data transferring between the cores. Additionally, the MIPS ISA defines the sample instructions to provide direct communication between the CPnCR and the GPR.

## *A. CUSTOM ISE*

Before defining the custom ISE for the core integration of Intellino, firstly, we set the identification number of the custom coprocessor to two. Next, we defined the custom ISE based on the register map of the original Intellino. Table 1 shows the instructions of the ISE. The instructions are descended from the sample move instructions and memory access instructions which are guided by the MIPS ISA

**TABLE 1.** **List of instructions on Intellino coprocessor.**

| | Instruction | Operands | Description |
|---|---|---|---|
| Data Transfer | MTC2 | rt, cs | $cs ← $rt |
| | MFC2 | rt, cs | $rt ← $cs |
| Memory Access | LWC2 | ft, offset(base) | $ft ← memory[base+offset] |
| | SWC2 | ft, offset(base) | memory[base+offset] ← $ft |

documentation. The main difference between the sample instructions and the ISE is the additional functions on write operations for the register file. The register map of the coprocessor is the same as the register map of the original Intellino. Hence, the instructions that contain writing the data to the CP2R, MTC2, and LWC2 perform certain operations similar to those of the neuron controller in the original. By this functionality, the mechanisms of the Intellino are conserved while the packet decoding and encoding process in the communication protocol is clearly eliminated.

Table 2 shows the information of the CP2Rs in the CP2 register file. Though almost every operation of the registers is the same as the original Intellino, some features are changed for optimization. The main difference between the original operations and newly defined operations is the default unit of the data transfers. Though the processor core relies on 32-bit operation, changing the 8-bit operations to 32-bit operation increases the throughput to four times faster than the original.

Basically, the training process and inference process of the Intellino are based on the iterative move operations that write the vector data to the address of the COMP register and LCOMP register, the MTC2 and LWC2. Writing the data to the COMP or LCOMP register in this state redirects to saving four component data, which is 32-bit, to the SPM in the neuron cell that has the same identifier as the NID value, addressed by the COMPID value. Further, every neuron cell performs the distance calculation for each based on the accumulation of the current distance value between the input component and the stored component. In case of the NID value exceeds the maximum identifier value ($2^n - 1$), nothing is stored in any neuron cell but the distance calculation is still executed. Consequently, the NID value indicates not

**TABLE 2.** Register map of the Intellino.

| Id. | Name | Op. | Description |
|---|---|---|---|
| 0 | IDLE | R | Resereved (always returns 0, do nothing) |
| 1 | COMP | W | Write vector component to the current neuron cell and increase the COMPID register value by four. |
| 2 | LCOMP | W | Write vector component to the current neuron cell and clear the COMPID register value to zero. If the current state is on inference state, updates the DIST register, and restart the process from the begining. |
| 3 | DIST | R | The distance between the recognition data and the closest neuron cell. |
| 4 | CAT | R/W | Read: Returns the category data recognized by the inference process. Write: Stores the category value and sets the valid flag in the current neuron cell and increases the NID value by one. |
| 5 | COMPID | R/W | Current index of the component in the vector for both the train state and inference state. |
| 6 | NID | R/W | Current index of the neuron cell to train. The NID exceeding the maximum identifier value ($2^n - 1$) indicates that the Intellino is in the inference state. |
| 7-13 | reserved |  | Reserved (always returns 0, do nothing ) |
| 14 | FORGET | W | Removes the trained data in the current neuron cell through clearing the valid flag and the COMPID register. |
| 15 | CLEAR | W | Resets the Intellino coprocessor. Forgets the entire datasets in the neuron cells and restart from the begining of the training process through clearing the NID and COMPID register. |
| 16-31 | reserved |  | Reserved (always returns 0, do nothing) |

```
// Load the vector data address to the certain GPR
lui  $at,    vec_addr_msb
ori  $at,    vec_addr_lsb
// Write the vector data to the neuron cell
lwc2 $comp,     0($at)
lwc2 $comp,     4($at)
...
lwc2 $lcomp, 124($at)
// Write the category value to the neuron cell
lwc2 $cat,   128($at)
```

**Listing 1.** Assembly codes for training based on the custom ISE.

only the neuron cell to train but also the current state of the whole coprocessor. Through this attribute, both the storing of the vector to the neuron cell in the training process and calculating the distance in the inference process are executed by the same iterative move operations. The variance between the training process and the inference process is that the training process ends with the writing operation of the category for training data to the neuron cell and the inference process ends with the reading operation of the derived category and distance. Another important attribute is that the operations to overwrite certain values to the COMPID register and NID register are highly infrequent because of the certain operations that automatically change the value in the register not addressed by the instruction, move operations for writing to the COMP, LCOMP, CAT, FORGET, CLEAR.

Listing 1 and 2 show examples of source code for the AI coprocessor executed by custom ISE and for the original Intellino executed by the SPI communication, respectively. Both source codes execute the training of single data from the dataset to the neuron cell. The length of the vector is configured as 128 and each of the COMPID values and the NID are regarded as respectively zero and the value lower than $2^n - 1$. Both codes share the steps that storing the training vector in the SPM through sequential writing and sending the category value. Despite the source codes having similar steps, the codes based on custom ISE provide much more simplicity than those based on the original AI processor due to the exclusion of the numerous function calls for the complex communications protocol.

```
training_data = (uint8_t*)vec_addr;
// Set data count to send the vector
iter_count = 0x007F;
// Call training operation
SPI.send(0x40);
// Send MSB 8-bit of the data count
SPI.send(iter_count >> 8);
// Send LSB 8-bit of the data count
SPI.send(iter_count & 0xFF);
// Write the vector data to the neuron cell
for (i = 0; i <= iter_count; i++)
    SPI.send(training_data[i]);
// Write the category value to the neuron cell
SPI.send(category);
```

**Listing 2.** The source codes for training based on the original architecture.

## V. SYSTEM ARCHITECTURE

Fig. 6 shows the proposed system architecture. The system consists of the AI ASIP for calculation, sensors for receiving inference data, and a display interface to interact with the users. The ASIP adopts the Harvard architecture, which has two paths, the data path and the instruction path, to avoid bubble insertion generated by sharing the same path on instruction fetching and memory access instructions. Compared to high-performance systems, which generally adopt the modified Harvard architecture, the exclusion of caches reduces area usage and energy consumption because of the elimination of the overheads caused by cache misses and avoiding the area occupancy by the cache. As the memory requirements of the lightweight AI are generous, the reduction of the available memory resources provided by the cache is acceptable. The integrated core on the ASIP processes the input data for inference by executing the program including the custom AI instructions through the internal AI coprocessor. The output controller displays the inference result calculated by the AI coprocessor, allowing the users to perceive the result directly. In accordance with this architecture, the throughput of the AI algorithm is improved compared to the systems based on the previous heterogeneous architecture. Further, the data flow of the

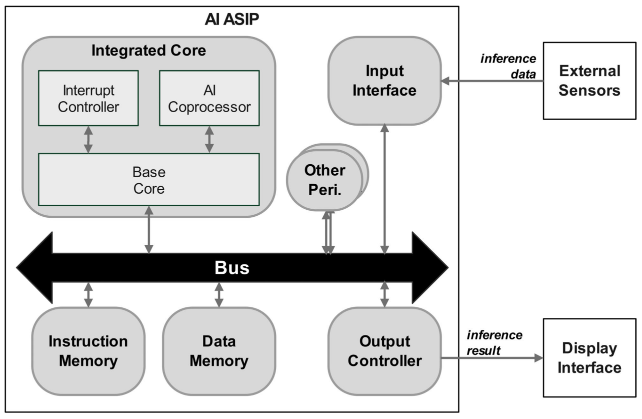

**FIGURE 6.** The proposed system architecture for lightweight AI systems.

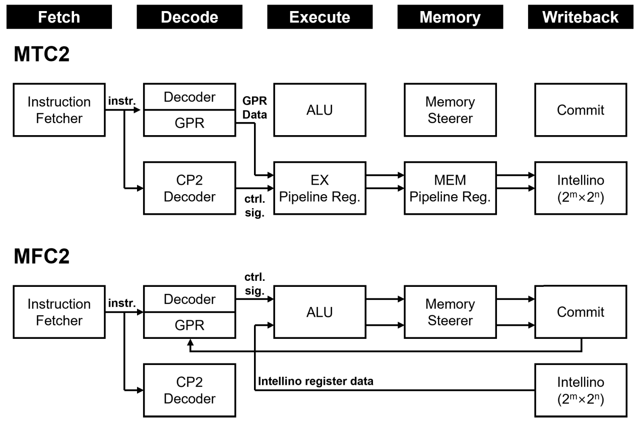

**FIGURE 7.** Data flow of the MTC2, MFC2 instructions on the core pipeline.

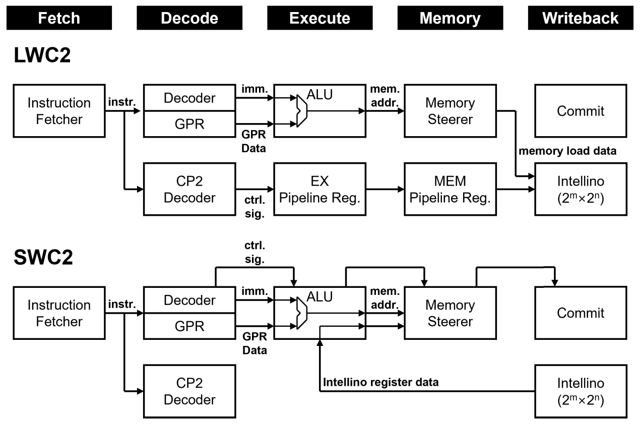

**FIGURE 8.** Data flow of the LWC2, SWC2 instructions on the core pipeline.

architecture provides privacy protection and network bandwidth reduction due to the edge computing architecture.

## A. INTEGRATED CORE DESIGN

To provide the ASIP based on MIPS ISA with custom AI ISE, we designed the AI coprocessor for a 5-stage pipelined MIPS core that is compatible with the 32-bit MIPS processor [42] and integrated the coprocessor into the MIPS core. The key aspects we considered in the design of the coprocessor are as follows:

- Conserving the logic related to the main operating mechanisms to ensure the functionalities of the legacy features that are enabled on the original AI processor.
- Optimizing the main calculation logic and memory architecture of the Intellino to the 32-bit parallel data transfer in the MIPS processor.

Fig. 7 and 8 show the data flow when executing the custom ISE instructions. The AI coprocessor, i.e., CP2, is designed to compose the coupled architecture with the pipelines in the basic MIPS core. The fetch stage in the pipeline is shared by both the basic core and CP2 as the instruction path is only one in the core. The other stages and pipeline registers for each stage exist for both the base core and the coprocessor to ensure synchronized operation.

As shown in the figures, the Intellino is held in the write-back stage. The stage to commit the 32-bit data to the Intellino register is the same as the time to commit to the GPRs. Through this architecture, the core integration of the Intellino is done without additional data forwarding logic.

The latency for AI instruction to reach the AI processor from the instruction fetch stage is five cycle periods of the operating clock frequency because of the pipelined architecture. In the training process, this characteristic does not generate throughput degradation as the pipeline operates without any interruption. In the inference process, in contrast, the throughput is reduced. This is because the data transfer instructions from the Intellino to the other units such as move from coprocessor 2 (MFC2) and store word from coprocessor 2 (SW2) complete the communication on the decode stage. To receive the appropriate inference results, the insertion of the five delay slots, which are the slots to fill with independent instructions next to the target instruction, is required.

Typically, methods to embed the additional forwarding logic are applied to remove the delay slots. Nevertheless, we adopt this architecture because adding the data forwarding logic directly affects the overall performance of the processor according to the base core architecture that data propagation of each stage passes through the data forwarding logic. In the worst case, the five no operation (NOP) instructions are inserted into every inference process. The throughput decrease by these bubbles is acceptable as the communication performance based on the operating frequency, which remains unchained due to the coprocessor architecture, is far ahead of the communication performance of the original heterogeneous architecture.

## B. AI CORE OPTIMIZATION

Fig. 9 shows the optimization of the distance calculation logic for 32-bit parallel communication. The distance calculation works by sequentially accumulating the absolute distance value between the current vector components addressed by

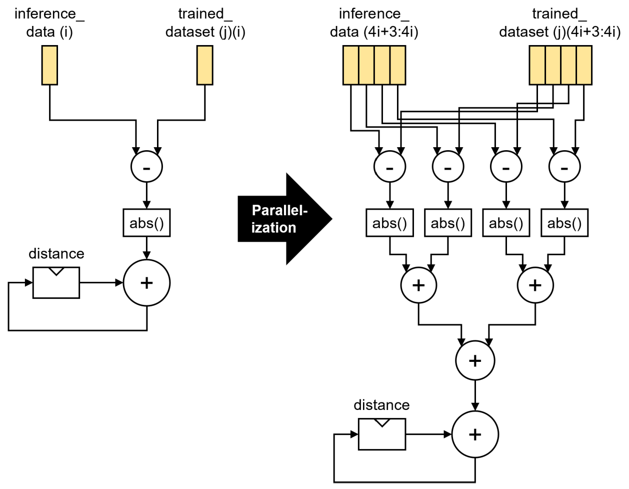

**FIGURE 9.** Optimization of the distance calculation for the 32-bit parallel communication.

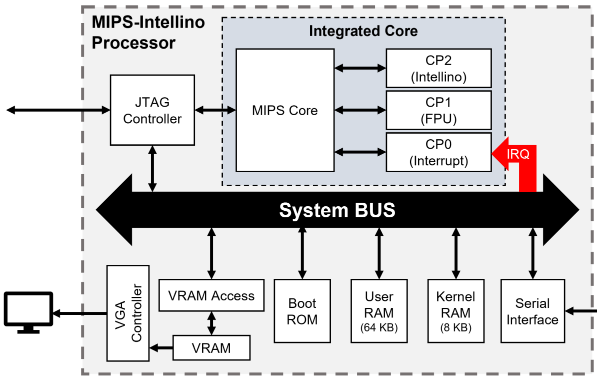

**FIGURE 10.** The architecture of the designed ASIP.

the COMPID register. In the case of the original AI processor, the distance calculation requires only one subtractor, absolute logic, and adder as the communication protocol unitizes the data packet to 8 bits. In the case of the AI coprocessor, in contrast, the communication is held with 32-bit parallel data wires. This means that the four vector components are simultaneously transferred to the calculation logic. To perform the distance calculation without additional bubble generation, we parallelized the sequential accumulations in the original intellino into four by duplicating the subtractor and absolute logic and adding the adders with a tree structure. With this architecture, the throughput of the optimized Intellino is improved as the distance is perfectly synchronized to the basic core pipeline.

### C. ASIP DESIGN

We designed the processor including the optimized AI coprocessor to realize and verify the concept of lightweight ASIP for AI applications. Fig. 10 shows the architecture of the processor. The processor including the integrated core architecture with interrupt controller (CP0), floating-point unit (CP1), and the designed AI core (CP2), enables the execution of the custom AI ISE and other features such as hardware interrupt and floating-point acceleration. The memory composition in Harvard architecture, which separates the instruction memory and data memory, is replaced by embedding the random access memories (RAM) with one read or write data path and one read-only data path, which imitates the mechanism that the instruction fetch stage and memory stage access the memory simultaneously. To provide reconfigurability of the memories, the connection between the master device of the system bus, the integrated core, and the system bus passes through the joint test action group (JTAG) controller. By sending the JTAG signals, the host device takes control of the system bus from the integrated core. This enables memory access from the host device to replace the current program with another. Moreover, the serial interface is provided to receive the training/inference data from the other input devices. Finally, the VGA controller and video RAM access peripheral are included to visualize the result.

## VI. IMPLEMENTATION RESULT

To verify the proposed system architecture, we first evaluated the processor designed to apply to the system by RTL simulation running on Verilator 5.009 version, which is an open-source RTL simulation tool. The simulation of the MIPS-Intellino processor has two cases: optimized assembly codes, and source codes with library functions written in C programming language. Both cases are built to operable machine codes by utilizing the GNU compiler collection (GCC) version 5.3.0 and supposed the operating frequency of the processor to be 50 MHz, which is a recommended frequency of the target MIPS processor and has not been modified. The simulation of the original Intellino has only one case, the SPI-based communication protocol with an 8 Mbps data rate, which is the recommended data rate of the Intellino.

We used the MNIST dataset [43], which is a test set of handwritten digits, for the k-NN training and inference process. The simulation environment and source codes for each case are provided on the website [44].

Fig. 11 shows the time measurement results of the training process. According to the simulation, the C library functions and optimized assembly codes showed approximately 165.59 times and 193.88 times enhanced performance compared to the original Intellino, which is based on SPI, on average. Both the throughput ratio of the operations compared to that of the original Intellino tends to converge to 200 when the vector length value increases. This is because the ideal throughput ratio between the 8 Mbps SPI communication and the 32-bit parallel communication running on 50 Mhz is 200 and the increase in the vector length does not inflate the ancillary instructions such as setting the starting pointer of the data, setting the category value. As a result, the throughput performance of the training process on MIPS-Intellino remarkably exceeded those of the original Intellino.

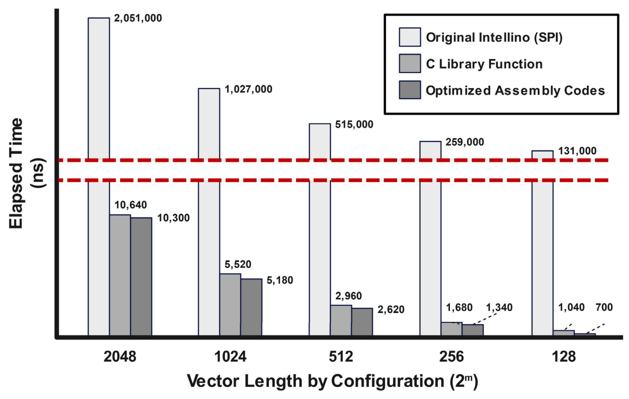

**FIGURE 11.** The required time for the training process in each case.

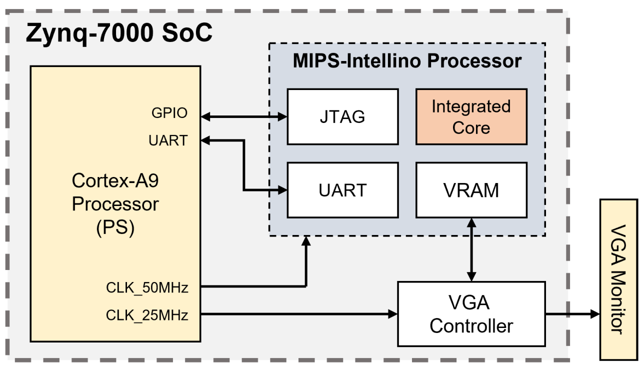

**FIGURE 12.** The architecture of the verification system.

After the simulation, we realized the designed processor by FPGA implementation. Fig. 12 shows the architecture of the verification system, which is based on the Zynq-7000 system-on-chip (SoC) mounted on the Digilent ZedBoard. The SoC consists of the general-purpose processor (PS) and the programmable logic (PL). The PS includes the complex block for the modern operating system (OS) such as the memory management unit (MMU) and a variety of peripherals. Certain peripherals, e.g., GPIO, in the PS is connectable to the programmable logic. This architecture facilitates the input/output process and the verification process of the designed digital circuit by making use of the convenient features the OS environment provides. Thus, we adopted the method of running the modern OS, Linux, on the PS and taking control of the design implemented on the PL block by the OS. The Linux distribution we utilized for the PS is PetaLinux 2021.1, and a detailed guide for running PetaLinux on the Zynq-7000 SoC is provided in [45].

To realize the method, firstly, we synthesized and implemented the MIPS-Intellino processor to the PL block of the Zynq-7000 SoC and linked the GPIO and the UART of the PS to the JTAG controller and the UART in the designed processor. Next, we developed the software that writes the program to the RAM in the designed processor by imitating the JTAG protocol through GPIO and programmed the test software compiled by the cross compiler built with the GNU C compiler (GCC) to the designed processor. At last, we checked the results of the test software running on the MIPS-Intellino through the serial communication software running on the PS.

To evaluate the area usage of the designed processor architecture, we performed synthesis and implementation using Xilinx Vivado 2022.2 for the designed coprocessor and the original AI processor with various AI configurations. The jobs were configured to target the PL block of the Xilinx xc7z020clg SoC with additional settings to avoid the utilization of pre-fabricated hard blocks, such as DSP blocks and block RAMs. Hard blocks in FPGAs provide performance enhancements and area reduction, operating more similarly to ASIC than FPGA [46]. Thus, utilizing these blocks is regarded as contracting the gap, e.g., area, power, and propagation delay, between ASICs and FPGAs [46]. However, in terms of purpose that uses the FPGA implementation as the prototyping of the digital logic circuit before ASIC implementation, utilizing these blocks makes the performance analysis fuzzier as the FPGA prototype operates as ASIC-FPGA hybrid architecture [47]. Therefore, to provide an approximately clear and comparative analysis on the different digital circuits for chip fabrication, synthesizing with only general-purpose blocks that is regarded as correspondence to the standard cell library is required.

Fig. 13 shows the resource usage of the implementation results between the previous AI processor and the designed AI coprocessor. The result presents that the FPGA implementations of the coprocessor require more look-up tables (LUTs) than those of the original AI processor. On average, the coprocessor uses approximately 6.62% more LUTs compared to the original. This is because the ancillary components of the coprocessor, the pipeline stages, use more LUTs than the ancillary components of the original processor, the SPI slave controller and the packet decoder. The gap between the LUTs usage of the coprocessor and the original decreases as the vector length parameter increases. As these components do not affect by the AI configurations such as vector length and the neuron cell count, the increase in the configuration parameters makes the area share on the AI logic and the memory higher, diminishing the impact of the gap between ancillary components. The usage of other physical resources, flip-flops (FFs), F7 multiplexers, and F8 multiplexers, of the coprocessor shows each of 74%, 40.72%, 19.53% reduced values than the original AI processor. According to this result, the area usage performance of the coprocessor is suitable for replacing the modularized architecture. When reflecting on the physical area reduction through the SoC architecture, which removes the external wires for communication, the advances in area usage performance become higher.

In the case of power and energy, the implementation results show that the designed AI coprocessor has far better performance than the original Intellino. Fig. 14 presents the energy-related performances of both designs. As shown in the first graph, the AI coprocessor consumes 3.58 times the dynamic power on average compared to the original. This

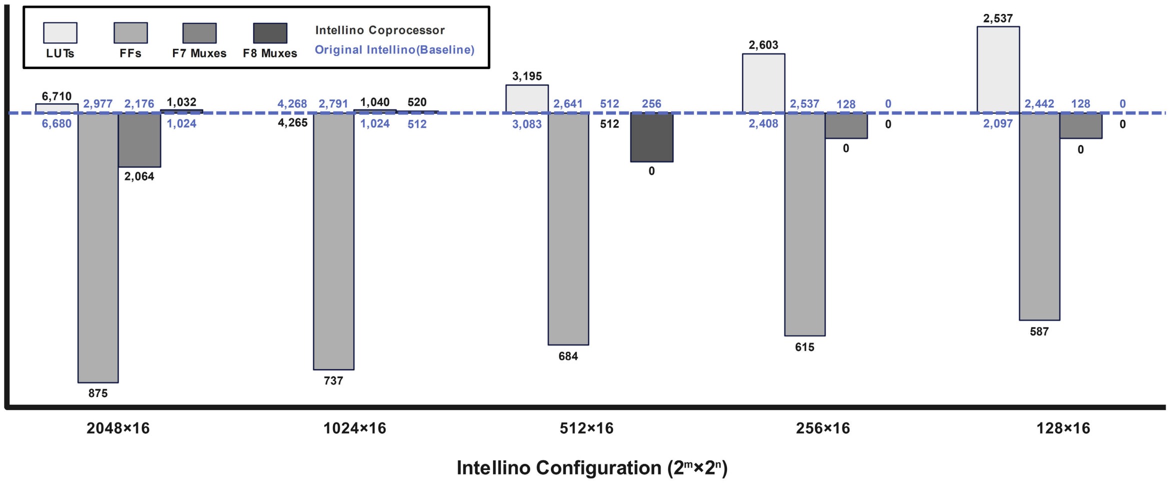

**FIGURE 13.** Resource usage comparison between the coprocessor and the original Intellino.

**TABLE 3.** Comparison between other k-NN accelerators.

| | This work | Intellino [40] | Curie PME [36] | CM1K [33] | NM500 [48] | [38] | [39] |
|---|---|---|---|---|---|---|---|
| Implementation | FPGA | FPGA | ASIC | ASIC | ASIC | FPGA | FPGA |
| Frequency (MHz) | 50 | $\leq$88 | 32 | 27 | 35 | 150.3 | 100 |
| Power (mW) | 123-136* | 109-112* | - | 15-275 | - | - | - |
| Vector length (byte) | $\leq$2048 | $\leq$4096 | $\leq$128 | $\leq$256 | $\leq$1024 | 8 | 2 |
| Data rate (MByte/s) | 182.91** | 0.95** | - | 24.98** | 31.15** | 12.50 | 12.28 |

*Measured in $n = 16$ configuration.
**Measured in $m = 8$ configuration.

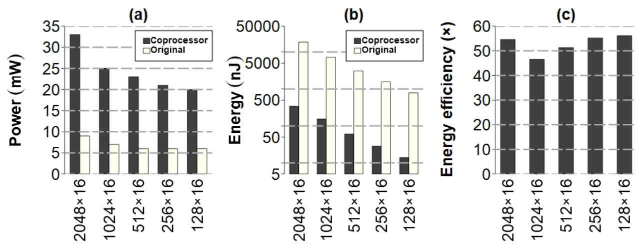

**FIGURE 14.** Power and energy comparison between the coprocessor and the original Intellino. (a) Maximum dynamic power consumption. (b) Energy dissipation per one operation. (c) Relative energy efficiency of the coprocessor compared to the original Intellino.

attribute results from the parallelized architecture consisting of the increment in memory bandwidth and the optimization of the distance calculation logic. The device static power statistics are the same for all configurations and designs, resulting in 0.103 W, as these attributes only depend on the target FPGA. Although the coprocessor requires more power than the standalone Intellino, the energy efficiency of the coprocessor surpasses the original, as shown in the second graph and third graph. This is because the energy dissipation (EDP) required for one operation is proportional to not only the power consumption but also the elapsed time for one operation such as the training process and inference process. As a result, the coprocessor shows 52.75 times the energy efficiency on average compared to the original, demonstrating the advances in energy efficiency.

Finally, we analyzed the timing reports of the coprocessor implementation. In spite of the increment in the propagation delay in the distance calculation logic issued by the parallelization, the recommended operating frequency of the target MIPS processor, 50 MHz, is preserved. This is because the increased delay is still less than the critical path of the basic core, 32-bit ALU operating with forwarded input. Thus, the performance of the basic instruction execution is preserved though the instruction set is extended by embedding the additional coprocessor for custom AI ISE. As a result, replacing the modularized AI coprocessor with the ASIP with integrated core architecture has benefits in both throughput and area usage while demeriting the flexibility of the system composition.

Table 3 presents a comparison between other k-NN accelerators designed for lightweight systems. The results show that the AI coprocessor we designed has significantly higher data rate performance than the others, while also offering a wide range of configurability in k-NN parameters and acceptable power consumption.

## VII. CONCLUSION

In this paper, we propose a uniprocessor system architecture based on ASIP for lightweight edge AI systems in order to reduce the communication overhead caused by the complex protocol and asynchronous operations of the heterogeneous architecture. To realize the proposed system architecture, we designed the ASIP supporting custom ISE based on the

MIPS ISA, which presents the guidelines for defining the custom ISE. The ISE we defined is optimized to mutate the target AI processor for heterogeneous computing, Intellino, a reconfigurable lightweight AI processor based on k-NN algorithm. The ASIP is designed with the integrated core architecture that contains the custom coprocessor that synchronously operates with the base core. By placing the core logic to execute the AI algorithm on the write back stage, the necessity of the additional forwarding logic is removed. Further, specified optimizations for communication which is based on multiple parallel wires, e.g., parallelization of the distance accumulation in the k-NN algorithm, boost the throughput remarkably with a little demerit in area usage performance.

According to the simulation results, the designed processor achieved up to 193.88 times enhanced throughput performance compared to that of the original modularized AI processor. In addition, resource shares of each component were reduced, except LUTs, which were 6.62% higher on average in compliance with implementation results. Considering the elimination of the external wires for communication due to the uniprocessor architecture is not reflected, the advantages of area usage are assumed to be higher. Therefore, adopting the proposed system architecture is reasonable for lightweight systems that have a fixed application and are hugely affected by the throughput.

In future work, we plan to advance our work through two separate stages. At first, we will eliminate the necessity of bubble insertion intrinsic in the current coprocessor architecture that reduces the throughput performance. This objective can be achieved by optimizing the microarchitecture such as reconstructing the data processing and propagation structure in the pipeline architecture and adding the data forwarding logic to avoid stalls. Second, we will adopt the proposed coprocessor architecture for other complex AI algorithms that are more complicated than k-NN, extending the proposed architecture for high-performance systems. Ultimately, we aim to fabricate the most advanced ASIP designed by future works onto a chip to prove the feasibility and performance enhancements of the proposed system architecture.

## REFERENCES

[1] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2018, pp. 63–71.

[2] S. Jang, H. W. Oh, Y. H. Yoon, D. H. Hwang, W. S. Jeong, and S. E. Lee, "A multi-core controller for an embedded AI system supporting parallel recognition," *Micromachines*, vol. 12, no. 8, p. 852, Jul. 2021. [Online]. Available: https://www.mdpi.com/2072-666X/12/8/852

[3] G. Cornetta and A. Touhafi, "Design and evaluation of a new machine learning framework for IoT and embedded devices," *Electronics*, vol. 10, no. 5, p. 600, Mar. 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/5/600

[4] J. Chen, M. Jiang, X. Zhang, D. S. da Silva, V. H. C. de Albuquerque, and W. Wu, "Implementing ultra-lightweight co-inference model in ubiquitous edge device for atrial fibrillation detection," *Expert Syst. Appl.*, vol. 216, Apr. 2023, Art. no. 119407. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417422024265

[5] P. Tsung, T. Chen, C. Lin, C. Chang, and J. Hsu, "Heterogeneous computing for edge AI," in *Proc. Int. Symp. VLSI Design, Autom. Test (VLSI-DAT)*, Apr. 2019, pp. 1–2.

[6] S. Mukhopadhyay, Y. Long, B. Mudassar, C. S. Nair, B. H. DeProspo, H. M. Torun, M. Kathaperumal, V. Smet, D. Kim, S. Yalamanchili, and M. Swaminathan, "Heterogeneous integration for artificial intelligence: Challenges and opportunities," *IBM J. Res. Develop.*, vol. 63, no. 6, pp. 4:1–4:1, Nov. 2019.

[7] C. Choi et al., "Reconfigurable heterogeneous integration using stackable chips with embedded artificial intelligence," *Nature Electron.*, vol. 5, no. 6, pp. 386–393, Jun. 2022, doi: 10.1038/s41928-022-00778-y.

[8] J. Han, M. Choi, and Y. Kwon, "40-TFLOPS artificial intelligence processor with function-safe programmable many-cores for ISO26262 ASIL-D," *ETRI J.*, vol. 42, no. 4, pp. 468–479, Aug. 2020, doi: 10.4218/etrij.2020-0128.

[9] D. Jamma, O. Ahmed, S. Areibi, G. Grewal, and N. Molloy, "Design exploration of ASIP architectures for the K-nearest neighbor machine-learning algorithm," in *Proc. 28th Int. Conf. Microelectron. (ICM)*, Dec. 2016, pp. 57–60.

[10] Y. Chen, H. Lan, Z. Du, S. Liu, J. Tao, D. Han, T. Luo, Q. Guo, L. Li, Y. Xie, and T. Chen, "An instruction set architecture for machine learning," *ACM Trans. Comput. Syst.*, vol. 36, no. 3, pp. 1–35, Aug. 2019, doi: 10.1145/3331469.

[11] H. W. Oh, J. K. Kim, G. B. Hwang, and S. E. Lee, "The design of a 2D graphics accelerator for embedded systems," *Electronics*, vol. 10, no. 4, p. 469, Feb. 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/4/469

[12] H. Huang, Z. Liu, T. Chen, X. Hu, Q. Zhang, and X. Xiong, "Design space exploration for YOLO neural network accelerator," *Electronics*, vol. 9, no. 11, p. 1921, Nov. 2020. [Online]. Available: https://www.mdpi.com/2079-9292/9/11/1921

[13] J. Wang and S. Gu, "FPGA implementation of object detection accelerator based on Vitis-AI," in *Proc. 11th Int. Conf. Inf. Sci. Technol. (ICIST)*, May 2021, pp. 571–577.

[14] E. Rapuano, G. Meoni, T. Pacini, G. Dinelli, G. Furano, G. Giuffrida, and L. Fanucci, "An FPGA-based hardware accelerator for CNNs inference on board satellites: Benchmarking with myriad 2-based solution for the CloudScout case study," *Remote Sens.*, vol. 13, no. 8, p. 1518, Apr. 2021. [Online]. Available: https://www.mdpi.com/2072-4292/13/8/1518

[15] M. Perotti, P. D. Schiavone, G. Tagliavini, D. Rossi, T. Kurd, M. Hill, L. Yingying, and L. Benini, "HW/SW approaches for RISC-V code size reduction," in *Proc. Workshop Comput. Archit. Res. RISC-V*, 2020, pp. 1–12.

[16] A. K. Verma, P. Brisk, and P. Ienne, "Rethinking custom ISE identification: A new processor-agnostic method," in *Proc. Int. Conf. Compil., Archit., Synth. Embedded Syst.*, New York, NY, USA, Sep. 2007, p. 125, doi: 10.1145/1289881.1289905.

[17] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovi, "The RISC-V instruction set manual. Volume 1: User-level ISA, version 2.0," California Univ. Berkeley Dept. Elect. Eng. Comput. Sci., Berkeley, CA, USA, Tech. Rep. UCB/EECS-2014-54, 2014.

[18] C. Price, "MIPS IV instruction set," Tech. Rep., Revision 3.2, 1995.

[19] A. S. Eissa, M. A. Elmohr, M. A. Saleh, K. E. Ahmed, and M. M. Farag, "SHA-3 instruction set extension for a 32-bit RISC processor architecture," in *Proc. IEEE 27th Int. Conf. Appl.-specific Syst., Archit. Processors (ASAP)*, Jul. 2016, pp. 233–234.

[20] T. Ahmed, N. Sakamoto, J. Anderson, and Y. Hara-Azumi, "Synthesizable-from-C embedded processor based on MIPS-ISA and OISC," in *Proc. IEEE 13th Int. Conf. Embedded Ubiquitous Comput.*, Oct. 2015, pp. 114–123.

[21] Y. Zhou, X. Jin, and T. Xiang, "RISC-V graphics rendering instruction set extensions for embedded AI chips implementation," in *Proc. 2nd Int. Conf. Big Data Eng. Technol.*, New York, NY, USA, Jan. 2020, p. 85, doi: 10.1145/3378904.3378926.

[22] M. Cococcioni, F. Rossi, E. Ruffaldi, and S. Saponara, "A lightweight posit processing unit for RISC-V processors in deep neural network applications," *IEEE Trans. Emerg. Topics Comput.*, vol. 10, no. 4, pp. 1898–1908, Oct. 2022.

[23] S. Lee, Y. Hung, Y. Chang, C. Lin, and G. Shieh, "RISC-V CNN coprocessor for real-time epilepsy detection in wearable application," *IEEE Trans. Biomed. Circuits Syst.*, vol. 15, no. 4, pp. 679–691, Aug. 2021.

[24] N. Wu, T. Jiang, L. Zhang, F. Zhou, and F. Ge, "A reconfigurable convolutional neural network-accelerated coprocessor based on RISC-V instruction set," *Electronics*, vol. 9, no. 6, p. 1005, Jun. 2020. [Online]. Available: https://www.mdpi.com/2079-9292/9/6/1005

[25] W. Lou, C. Wang, L. Gong, and X. Zhou, "RV-CNN: Flexible and efficient instruction set for CNNs based on RISC-V processors," in *Advanced Parallel Processing Technologies*, P.-C. Yew, P. Stenström, J. Wu, X. Gong, and T. Li, Eds. Cham, Switzerland: Springer, 2019, pp. 3–14.

[26] Z. Li, W. Hu, and S. Chen, "Design and implementation of CNN custom processor based on RISC-V architecture," in *Proc. IEEE 21st Int. Conf. High Perform. Comput. Commun., IEEE 17th Int. Conf. Smart City, IEEE 5th Int. Conf. Data Sci. Syst.*, Aug. 2019, pp. 1945–1950.

[27] R. Porter, S. Morgan, and M. Biglari-Abhari, "Extending a soft-core RISC-V processor to accelerate CNN inference," in *Proc. Int. Conf. Comput. Sci. Comput. Intell. (CSCI)*, Dec. 2019, pp. 694–697.

[28] E. Flamand, D. Rossi, F. Conti, I. Loi, A. Pullini, F. Rotenberg, and L. Benini, "GAP-8: A RISC-V SoC for AI at the edge of the IoT," in *Proc. IEEE 29th Int. Conf. Appl.-Specific Syst., Archit. Processors (ASAP)*, Jul. 2018, pp. 1–4.

[29] F. Taheri, S. Bayat-Sarmadi, and S. Hadayeghparast, "RISC-HD: Lightweight RISC-V processor for efficient hyperdimensional computing inference," *IEEE Internet Things J.*, vol. 9, no. 23, pp. 24030–24037, Dec. 2022.

[30] J. Kim, J.-K. Kang, and Y. Kim, "A resource efficient integer-arithmetic-only FPGA-based CNN accelerator for real-time facial emotion recognition," *IEEE Access*, vol. 9, pp. 104367–104381, 2021.

[31] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size," 2016, *arXiv:1602.07360*.

[32] M. Xia, Z. Huang, L. Tian, H. Wang, V. Chang, Y. Zhu, and S. Feng, "SparkNoC: An energy-efficiency FPGA-based accelerator using optimized lightweight CNN for edge computing," *J. Syst. Archit.*, vol. 115, May 2021, Art. no. 101991. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762121000138

[33] General Vision. (Aug. 2017). *CM1K Hardware User's Manual*. Accessed: Apr. 8, 2023. [Online]. Available: https://www.generalvision.com/documentation/TM_CM1K_Hardware_Manual.pdf

[34] L. L. Abeysekara and H. Abdi, "Short paper: Neuromorphic chip embedded electronic systems to expand artificial intelligence," in *Proc. 2nd Int. Conf. Artif. Intell. Industries (AI4I)*, Sep. 2019, pp. 119–121.

[35] M. Suri, V. Parmar, A. Singla, R. Malviya, and S. Nair, "Neuromorphic hardware accelerated adaptive authentication system," in *Proc. IEEE Symp. Ser. Comput. Intell.*, Dec. 2015, pp. 1206–1213.

[36] R. Dower. (2022). *IntelB. Pattern Matching Technology*. Accessed: Apr. 8, 2023. [Online]. Available: https://github.com/intel/Intel-Pattern-Matching-Technology

[37] A. Borelli, F. Spagnolo, R. Gravina, and F. Frustaci, "An FPGA-based hardware accelerator for B the B $k$-nearest neighbor algorithm implementation in B wearable embedded systems," in *Applied Intelligence and Informatics*, M. Mahmud, C. Ieracitano, M. S. Kaiser, N. Mammone, and F. C. Morabito, Eds. Cham, Switzerland: Springer, 2022, pp. 44–56.

[38] H. Hussain, K. Benkrid, C. Hong, and H. Seker, "An adaptive FPGA implementation of multi-core K-nearest neighbour ensemble classifier using dynamic partial reconfiguration," in *Proc. 22nd Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2012, pp. 627–630.

[39] Z.-H. Li, J.-F. Jin, X.-G. Zhou, and Z.-H. Feng, "K-nearest neighbor algorithm implementation on FPGA using high level synthesis," in *Proc. 13th IEEE Int. Conf. Solid-State Integr. Circuit Technol. (ICSICT)*, Oct. 2016, pp. 600–602.

[40] Y. H. Yoon, D. H. Hwang, J. H. Yang, and S. E. Lee, "Intellino: Processor for embedded artificial intelligence," *Electronics*, vol. 9, no. 7, p. 1169, Jul. 2020. [Online]. Available: https://www.mdpi.com/2079-9292/9/7/1169

[41] D. H. Hwang, C. Y. Han, H. W. Oh, and S. E. Lee, "ASimOV: A framework for simulation and optimization of an embedded AI accelerator," *Micromachines*, vol. 12, no. 7, p. 838, Jul. 2021. [Online]. Available: https://www.mdpi.com/2072-666X/12/7/838

[42] H. W. Oh, K. N. Cho, and S. E. Lee, "Design of 32-bit processor for embedded systems," in *Proc. Int. SoC Design Conf. (ISOCC)*, Oct. 2020, pp. 306–307.

[43] L. Deng, "The MNIST database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 141–142, Nov. 2012.

[44] H. W. Oh. (2023). *Simulation ENV. for The MIPS-Intellino and the Original Intellino*. Accessed: Apr. 9, 2023. [Online]. Available: https://github.com/hopesandbeers/MIPS-Intellino-sim

[45] Xilinx. (2022). *ZYNQ-7000 Embedded Design Tutorial*. Accessed: Nov. 8, 2022. [Online]. Available: https://xilinx.github.io/Embedded-Design-Tutorials/docs/2021.1/build/html/docs/Introduction/Zynq7000-EDT/Zynq7000-EDT.html

[46] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 2, pp. 203–215, Feb. 2007.

[47] A. Ehliar and D. Liu, "An ASIC perspective on FPGA optimizations," in *Proc. Int. Conf. Field Program. Log. Appl.*, Aug. 2009, pp. 218–223.

[48] General Vision. (Apr. 2019). *NM500 User's Manual*. Accessed: Apr. 8, 2023. [Online]. Available: https://generalvision.com/documentation/TM_NM500_Hardware_Manual.pdf

**HYUN WOO OH** received the B.S. degree in electronic and IT media engineering and the M.S. degree in electronic engineering from the Seoul National University of Science and Technology, Seoul, South Korea, in 2021 and 2023, respectively. He has published some papers and articles related to processor architecture and posit arithmetic. His current research interests include computer architecture, system-on-chip, and edge computing.

**SEUNG EUN LEE** (Senior Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, in 1998 and 2000, respectively, and the Ph.D. degree in electrical and computer engineering from the University of California, Irvine (UC Irvine), in 2008. After graduating, he had been with Intel Labs., Hillsboro, OR, USA, where he worked as a Platform Architect. In 2010, he joined as a Faculty Member with the Seoul National University of Science and Technology, Seoul. His current research interests include computer architecture, multi-processor system-on-chip, low-power and resilient VLSI, and hardware acceleration for emerging applications.