Journal of Innovations in Computer Science and Engineering
Shahid Beheshti Unversity

May 2023, Volume 1, Issue 1

NIMA: A NISC-based Micro Architecture Design Methodology for Fog Computing Applications

Somayeh Maabi [a] Code Orcid: 0000-0002-7126-7381,
Seyed-Hosein Attarzadeh-Niaki[b] Code Orcid: 0000-0002-2171-1528,,
Maghsoud Abbaspour[c] ✉ Code Orcid: 0000-0002-7126-7381
[a] Faculty of Computer Science and Engineering, Shahid Beheshti University, Tehran, Iran, s_maabi@sbu.ac.ir
[b] Faculty of Computer Science and Engineering, Shahid Beheshti University, Tehran, Iran, h_attarzadeh@sbu.ac.ir
[c] Faculty of Computer Science and Engineering, Shahid Beheshti University, Tehran, Iran, maghsoud@sbu.ac.ir

ABSTRACT

With the growing adoption of Internet of Things (IoT) technologies, the number of connected devices and complexity of applications are increasing. To address the challenge of handling large amounts of data, novel concepts such as edge and fog computing have been proposed, introducing processing, connectivity, and storage capabilities between devices and the cloud. However, designing these layers introduces new challenges, especially at the architecture and micro-architecture levels. This article introduces NIMA, a systematic NISC-based (No Instruction Set Computer) micro-architecture design methodology that enables rapid and optimal customization of a processor architecture for a specific application domain, utilizing a representative benchmark. To optimize the processor data-path, a novel utilization metric and heuristic algorithm are introduced. Our methodology's effectiveness is demonstrated by designing processors that optimize performance, area, or power objectives for a proposed benchmark in the fog computing domain. Experimental results show significant improvements, particularly 14.5% in performance and 9.2% in power, using the proposed NIMA_PF and NIMA_PW architectures compared to a conventional MIPS-based

KEYWORDS

Edge computing, microprocessor design methodology, fog computing, micro-architecture customization, IoT devices.

1. INTRODUCTION

The Internet of Things (IoT) is composed of interconnected and uniquely identifiable physical devices that collect data and enhance productivity, aiming to reduce or eliminate human intervention in data acquisition, interpretation, and use. With billions of devices estimated for IoT use-cases, such as healthcare, smart cities, smart homes, transportation, and manufacturing, transmitting vast amounts of data results in significant increases in costs, including power consumption and latency [1], [2] and [3]. To address this challenge, edge and fog computing are introduced, performing computations at end or edge nodes while fog extending cloud computing to the network's edge. These computing models provide compute, storage, and networking services, minimizing data transmission and congestion [3], [4] and [5]. Fog/edge computing is critical for processing data close to the source, necessitating an efficient computing platform for fog devices [6], [7]. IoT microprocessors, particularly fog microprocessors, require domain-specific designs compared to conventional general-purpose computing architectures [8]. The limitations of available embedded processors pose significant challenges for their use as fog nodes [9], requiring further research into micro-architectural optimization. This optimization allows designers to develop right-provisioned architectures, which are efficient, configurable, extensible, and scalable enough for next-generation IoT devices [3].

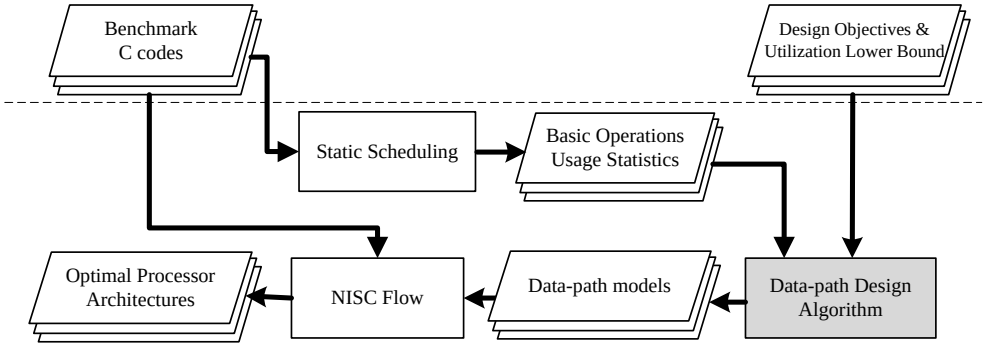

Figure 1. The proposed systematic design flow

Submit Date: 2023-05-03
Revise Date: 2023-07-18
Accept Date: 2023-10-13
✉ Corresponding author

We propose a systematic design methodology based on the No-Instruction-Set Computer (NISC) concept to explore the processor design space for fog computing applications, with a focus on data-path elements and micro-architecture (Figure 1). The methodology starts with a representative benchmark for the target domain, which is prepared based on a thorough study of published literature (Section 5.1). We introduce a novel utilization parameter to guide the design flow towards choosing the relative abundance of data-path modules for the micro-architecture (Section 4.1). Once the lower bound on the utilization parameter is decided, other design parameters are deduced based on the intended design objectives to achieve an optimal data-path. A static scheduling is performed on the benchmark algorithms' intermediate representation (Section 4.2), followed by the optimization of the basic input processor's data-path using a proposed heuristic algorithm (Section 4.3). Finally, the NISC tool-set is used to design the processor architecture based on the optimized data-path, and the applications are compiled. We compare the performance and cost of the proposed processors with a comparable RISC processor implementation for the suggested benchmark (Section 5.5.2).

To avoid lengthy design iterations, we propose a set of initial seeds for design parameters based on experimental evidence. This approach enables faster convergence to an optimal design considering design objectives instead of time-consuming full design-space exploration. The proposed systematic design flow can be applied by the designer to optimize other processor families with any desired design objectives. The main contributions of this article are mentioned below.

- Introducing NIMA, a systematic design methodology (Figure 1) that follows the NISC design philosophy. This approach customizes the micro-architecture of the processor to optimize performance and power for a particular application domain benchmark, as outlined in Section 4.
- Introducing a novel utilization metric in Section 4.1, which, in conjunction with a lower bound demonstrated experimentally in Section 5.2, is used to derive the optimal quantity of elements in the data path (Section 5.4).
- Employing a heuristic algorithm with proposed parameters based on objectives to design a data-path that achieves optimal performance, area, and power consumption, as illustrated in Figure 1. Section 4.3 provides a detailed exposition of this methodology and demonstrated experimentally for the benchmark in Section 5.3.
- Designing sample processor architectures with optimum performance and cost for a fog-computing benchmark (Section 5).

2. RELATED WORK

We present a review of the state-of-the-art articles in three domains. The first section covers microprocessor optimization and system on chip (SoC) design for IoT applications with an emphasis on performance metrics. The subsequent section delves into NISC-based system design articles and discusses the related research for IoT in detail. While many papers offer network architectures for fog computing services and underlying theory [10], they pay little attention to the challenges of fog computing for embedded system design. Thus, in the final subsection, we examine papers related to fog computing.

2.1. RISC-Based IoT Processor Design

Gautschi et al. [11] proposed a reconfigurable RISC processor for IoT applications that reduces pipeline stages to two, decreasing power consumption, chip area, and simplifying the design. However, the authors note that this architecture is not suitable for all applications and is specifically tailored to those with low performance requirements. The development of the heterogeneous and energy-efficient dual-core processor, coreLH [12], addresses the challenge of maximizing performance while meeting task deadlines in IoT applications. The coreLH architecture enables data sharing between power domains and optimizes task scheduling for low energy consumption, achieving energy efficiency at lower voltages due to its higher maximum operating frequency compared to similar designs. Huan et al. [13] presented a system-on-chip (SoC) implementation for the IoT domain featuring a reconfigurable micro-coded application-specific instruction-set processor (ASIP). This SoC provides a general-purpose control mechanism to accelerate specific applications through a reconfigurable ASIP core. Also it utilizes two types of network access for analog and digital sensors, demonstrating energy efficiency and area savings just for two IoT application algorithms. Results demonstrated an increase in core speed for sensor processing workloads with compressed data.

Several other studies [14], [15], [16] and [17] have focused on improving power consumption, flexibility, performance, and memory management in IoT end or edge node devices. Some of these studies present ASIP or ASIC designs for specific applications, such as the ASIP design proposed by [18] for automotive applications. The SenseASIP platform features a microprocessor architecture with customized instructions for computing numerous signal processing tasks in the sensor.

2.2. NISC-Based IoT Processor Design

In addition to RISC-based processors, novel solutions employing the NISC approach have been proposed for achieving optimal power and performance in IoT applications, as demonstrated in [19], [20], [21], [22] and [9]. The NISC architecture offers data-path configurability and programmable control, providing flexibility and reducing time-to-market while achieving optimal performance and power consumption [22]. Some of the reviewed articles in this category present processor designs specifically tailored for IoT applications, which are discussed below.

Efficient implementation of NISC architecture for IoT applications with low complexity has proven to be effective by Rizk et al. As the complexity and variety of IoT applications continue to increase, and new digital connection standards are continuously developed, they recognized the need for a flexible architecture with different transferring component models [19]. Their objective was to offer an optimized, flexible architecture for supporting various wireless connection standards (e.g., LTE, Wi-Fi, WiMAX, and DVB-RCS) in a receiver system for IoT applications. To this end, they presented an application-specific processor with NISC architecture and design flow called MIMO Turbo-equalizer, which demonstrated superior performance and flexibility. In [19], Rizk proposed the Minimum Mean-Squared Error algorithm to improve the error rate calculation performance of the previously mentioned equalizer, which had considerable complexity. They also designed a processor for the Minimum Mean-

Squared Error Interface Canceller (MMSC-IC) turbo equalizer [19]. The results showed a significant improvement in throughput with reduced implementation costs.

To address the decoding overhead of automatic scheduling in the ASIC concept, a new de-mapper system with the NISC concept was introduced in [21]. The study presented a NISC design flow for developing a public de-mapper for several wireless standards. Compared to an implemented ASIP model, the NISC model's de-mapper overcomes architectural challenges with reduced implementation costs and requires less memory for implementing the control memory. Furthermore, results show significant improvements in running time and area with the same computing resources [21].

2.3. System Design for Fog Computing

Zeng et al. [23] discussed resource management in software-based embedded systems for fog computing, including task scheduling and placement of task images on storage servers. The authors addressed the challenges faced by popular embedded systems and addressed three topics: time scheduling, resource management, and adjusting I/O interrupt requests between storage servers. Their paper focused on a software-defined embedded system for supporting fog computing applications. They formalized these items as a mixed-integer nonlinear programming problem and validated them through extensive simulation examinations. In [24], Brzoza et al. proposed four hardware platform design principles for fog computing applications in environmental monitoring. The authors studied the requirements and challenges of distributed data acquisition systems and discussed the feasibility of deploying different versions of the platform for these applications. They tested the developed prototypes in a level monitoring use case.

Table 1, compares the reviewed articles on IoT custom processor or system design with the proposed architecture in this article, namely the NIMA processor, which presents a design methodology based on desired objectives, including parametric methods for processor design. The focus is on the control word and data-path, rather than the ISA. The paper proposes a design methodology for fog computing applications with specified design objectives. A key feature of the proposed architecture is the ability to completely change the data-path based on the specified objectives during the design phase, which, along with the absence of an ISA, simplifies the design flow and reduces time to market. It should be noted that the proposed design flow is not limited to application-specific design but is also applicable in a benchmark-specific context. Despite having a longer time to market window than general-purpose designs, the proposed design flow offers significantly higher flexibility and efficiency.

Table 1. Comparison of related projects in the IoT domain with our work.

Articles	Optimization Approach	Implementation Arch	Hints
[11]	Power,Area	ASIP, RISC	Low performance apps
[12]	Performance	Dual-core processor	Not energy efficient
[13]	Energy, Area	SoC (ASIP processor (RISC), network accessing)	limited efficiency
[14],[15], [16],[17], [18]	Power, Performance	ASIC, ASIP/ RISC	Application specific
[19],[20], [9],[21]	Performance, Power	ASIC/ NISC	No ISA
[23]	-	SoC	Software defined
[24]	Performance, Power, Area	Hardware platform design	Application specific
NIMA processor	Performance, Power, Area	Domain specific IP/ NISC	No ISA, Parametric design flow

3. BACKGROUND: THE NO-INSTRUCTION-SET COMPUTER

This section introduces the concept of No-Instruction-Set Computer (NISC) and its associated design methodology (Figure 2), which forms the basis of the proposed processor architecture. Two related approaches for designing processing elements are Application-Specific Instruction-Set Processors (ASIP) and High-Level Synthesis (HLS) [26]. However, HLS algorithms target a specific application and lack the flexibility to support a range of applications [25]. In contrast, ASIPs require a special data path and an instruction refinement, decoder, and compiler, limiting the number of custom instructions and resulting in complex tasks that require particular expertise [27]. The NISC concept eliminates the instruction-set abstraction by translating programs directly to a processor data-path, allowing designers to focus on the system description and data-path design, rather than designing the controller. NISC generates the RTL model of the processor and control words for the desired application in the NISC design methodology, which can be implemented in the target technology [28]. The NISC tool-set facilitates this flow and combines the best of both general-purpose processors and custom hardware design [27].

3.1. The NISC Architecture

A NISC processor architecture comprises a pipelined data-path and controller that drive the control signals of the data-path components at each clock cycle [27]. The data-path is fine-tuned statically by adding or eliminating components and their interconnection, and can range from a simple RISC to a complex data-path of a high-end processor [27]. To model the data-path of a NISC processor, the designer can use an Architecture Description Language (ADL) called Generic Netlist Representation (GNR), or opt for automated approaches such as HLS tools or a library of standard, pre-designed architectures.

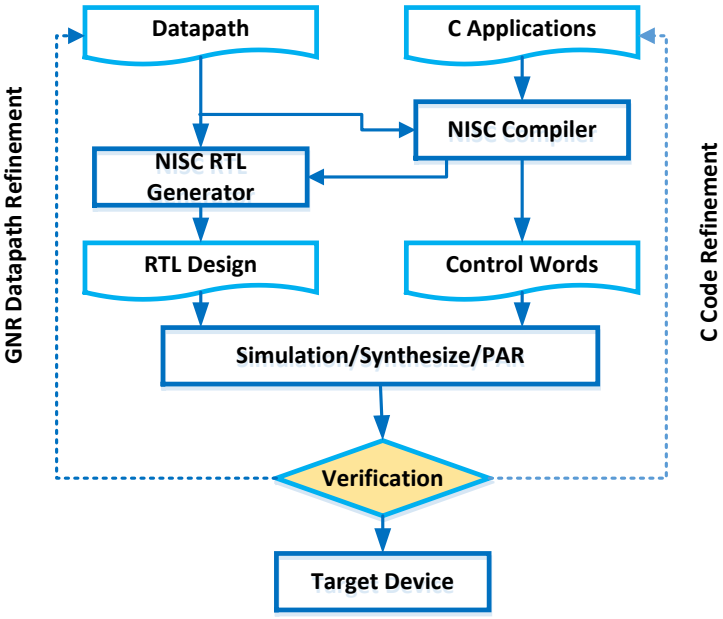

Figure 2. The NISC design flow

4. NIMA: THE PROPOSED SYSTEMATIC DESIGN FLOW

In custom processor architecture design, data-path design is a crucial step [20]. To obtain an optimal data-path and, consequently, an optimal processor architecture for a domain-specific benchmark, we propose a six-step systematic design flow (Figure 1). At first we introduce the design metric and a set of design parameters used in our design flow. Next, we explain the three key activities in the design flow, depicted as rectangles in Figure 1. The proposed flow is further clarified by applying it to a fog computing benchmark in Section 5.

4.1. The Utilization Metric and Design Parameters

We introduce *utilization* as a metric to measure the usefulness of a functional unit (FU) in the data-path for a given benchmark. It is used as a means to determine and adjust other design parameters such as register file (RF) port size and data-path connections to optimize the processor performance, area and power consumption.

Assume that a function f is scheduled using basic operations, corresponding to a set of given functional unit types, without resource constraints. Such a scheduling is typically performed on the intermediate data flow graph of functions and are a common pass in high-level synthesis tools, which are explained further in Section 5. For such a schedule, which executes in c_f cycles and for each functional unit \$u\$ of its data-path, the *concurrency level function* $n_{u,f} : N^+ \rightarrow R$ is defined such that $n_{u,f}(l)$ returns the number of cycles where l instances of u (each u related to an instruction) are scheduled concurrently. The *utilization function* $\mu_{u,f} : N^+ \rightarrow R$, which normalizes $n_{u,f}$ for the execution length c_f, is defined as below.

$$\mu_{u,f}(l) = \frac{n_{u,f}}{c_f} \qquad (1)$$

For a benchmark of M, with different applications (functions) $M = \{f_1, \dots, f_F\}$, we define the \emph{mean utilization function} $\mu_{u,M} : N^+ \rightarrow R$ as the following.

$$\mu_{u,M}(l) = \frac{\sum_{f=1}^{F} \mu_{u,f}}{F} \qquad (2)$$

To account for varying cycle lengths during scheduling and equalize utilization effects across all cycles of the benchmark, we employ normalization based on cycles followed by averaging. Plotting $\mu_{u,M}$ for different levels of concurrency 1,2, ... yields the *utilization histogram* of u for benchmark M. Once the designer decides on a lower bound, μ_{lb}, for the utilization of concurrency levels,

$$n_{u,M,\mu_{lb}} = \min\left(1, \max\{l | \mu_{u,M}(l) \geq \mu_{lb}\}\right) \qquad (3)$$

, determines the exact number of each functional unit \$u\$ in the data-path, which is the maximum level of concurrency that satisfy the lower bound on utilization. Section 5.2 presents a heuristic for deriving a utilization lower bound to optimize the data-path. While a single utilization lower bound for all functional units can be considered, a separate utilization lower bound for each functional unit provides better resolution for optimization. The desired utilization lower bound (μ_{lb}) is determined by

the designer, and the optimal number of functional units is then determined accordingly. The utilization function is used to link the number of each functional unit with the design objectives (e.g., performance and cost). The number of instances of each functional unit in the data-path depends on its degree of utilization of the corresponding basic operation in the schedule. At least one instance of each basic functional unit is included in the data-path, while extra instances are added as needed to satisfy the lower bound on utilization (μ_{lb}) in the schedule (Equation 3).

4.2. Extraction and Analysis of Data-path Elements

A processor design consists of two levels: instruction-set architecture and micro-architecture. This article focuses on the micro-architecture level, proposing a semi-automated method in Section 3.1 to determine the type and number of data-path elements. Our method utilizes basic-block scheduling and profiling information from benchmark applications to design the data-path. The profiling information can be inferred from call graphs generated by the static scheduling pass of typical HLS flows. The back-end pass of HLS flow contains four steps: allocation, scheduling, binding, and generating RTL [32]. The scheduling step yields the application control flow graphs, and the binding step instantiates the functional units where LLVM instructions are executed. Usage statistics and the simultaneity of FUs in scheduled cycles are obtained in the HLS flow.

The extracted information for each FU is analyzed based on the utilization metric (Section 4.1) to determine the utilization distribution of the FUs. A concurrency level histogram for each FU is then created to determine the exact number of FUs (explained in Section 5.2). This histogram displays the distribution of concurrency levels for each FU, as determined by the number of clock cycles scheduled for benchmark algorithms. For each design, μ_{lb} is the lower bound on the average simultaneous usage of FUs in different cycles, which specifies the performance, area, and power criteria for the best selection of FU numbers on the histogram. The evaluation section presents the optimum number of FUs for the benchmark algorithms (Table 5).

input : FUs types, Performance, Power and Area constraints, μ_{lb}, $\mu_{u,M}$
output: number of FUs, RF ports size, number of Pipeline registers and forwarding links

for *(each FU type in benchmark algorithms)* **do**
  Calculate the exact value of FU ($n_{u,m,\mu_{lb}}$) based on μ_{lb};
end

// *RF port size*
if *(objective = performance)* **then** (RF port size = RF_d);
else if *(objective = area)* **then** (RF port size = RF_a);
else (RF port size = RF_p);
if *(RF port size* $<$ $\max(n_{u,m,\mu_{lb}})$*)* **then** (reuse RF ports for the same FUs);
for *(each FU type in the data-path)* **do**
  Assign the RF ports to the FU ports in Round Robin fashion;
end
// *pipeline registers*
if *(objective = performance)* **then** (Reg numbers = R_d);
else if *(objective = area)* **then** (Reg numbers = R_a);
else (Reg numbers = R_p);
// *forwarding path*
Establish forward links in the data-path

Algorithm 1: The proposed date-path design algorithm

4.3. *The Data-path Design Algorithm*

To design a data-path, appropriate tuning of its features, such as the type and number of functional units (e.g., ALU, adder, subtractor, etc.), size of the RF ports, data-path forwarding links, and pipeline registers, is required, depending on the desired design objective(s). Our proposed objective-based data-path design algorithm applies several parametric customization, based on the introduced utilization metric, to achieve optimal data-paths. In a previous paper [30], we proposed two data-path design methods with optimal performance, using exhaustive search to find the optimum data-path with specified constraints on performance and power. In this work, we present a parametric design algorithm, introduced in Algorithm 1, to customize and design optimal data-paths based on the target. Input parameters for the algorithm include the extracted utilization statistics for FUs, $\mu_{u,M}$ and μ_{lb}, as well as performance, power consumption, and area constraints, typically assigned by system architects in the early stages of the processor design flow. The utilization parameter serves as a road map throughout the processor architecture design. The required number of FUs ($n_{u,M,\mu_{lb}}$) is first determined using the proposed lower bound μ_{lb} on the utilization values ($\mu_{u,M}$). This is followed by parametric customization of the data-path considering the calculated value of FUs ($n_{u,M,\mu_{lb}}$), required

simultaneity of the scheduled operations, and the target design objective. The RF port size, pipeline registers, and forwarding links are then determined to support the required operation parallelism in the processor. The RF port size is specified depending on the performance, area, or power objectives. The forwarding technique, in addition to reducing the execution time, allows for the best level of concurrency and optimal power consumption in case of connection with the pipeline registers. Pipeline registers and forwarding paths are established to increase simultaneity and address the need for more efficiency. These parameters are introduced heuristically (Table 7), but are proven appropriate by empirical evidence (Table 6) in Section 5.3.

4.4. Applying The NISC Flow: Final Processor Design

Using the NISC flow, the processor design is finalized by selecting data-path modules from the NISC tool-set libraries based on the design algorithm. A GNR file with an XML structure describes the data-path design, which is then imported along with benchmark algorithms to design the optimal architecture. The architecture is synthesized and simulated using standard hardware design tools to measure performance and costs. If the design meets the specified constraints, it is finalized as an optimal processor; otherwise, the data-path must be redesigned.

5. EVALUATION WITH A FOG COMPUTING BENCHMARK

In the evaluation phase, the proposed design flow is validated by a suggested benchmark, which represents typical fog applications, with three different design objectives. The three main steps of the design flow, as well as their input and output files, are explained in detail after describing the proposed benchmark.

5.1. A Representative Benchmark for Fog Computing

After studying the literature and applications on fog computing, the main operations and infrastructure requirements of a fog node are inferred from published articles [10], [31], [32], [33] and [34],

- The application domains of fog computing; and
- Taking into account the fog node's position in the network hierarchy.

Two viewpoints in application domains of fog computing are, fog applications and services that are not efficiently supported by cloud computing, such as geographically distributed applications for environmental monitoring, fast mobile applications like smart connected vehicles, and distributed control systems, are considered [10]. Additionally, applications that benefit from fog computing, like healthcare, smart grid, smart vehicles, emergent computing, and augmented reality applications, are also taken into account [31], [34].

The network hierarchy model [35] is used to express the requirements and operations associated with fog applications. The type and density of operations of a fog node depend on its distance from the end node or data center. Nodes closer to the end node focus on data gathering, normalization, and management of sensors and actuators. As the distance from the edge node increases, the focus shifts towards data filtering, compression, transformation, and analytical capabilities based on machine learning [35]. Depending on the fog use-case, applications may be memory or compute-intensive with varying run-time requirements [17]. Table 2 summarizes the main operations, basic algorithms, and examples of applications for each requirement type. This work specifically targets the acceleration of computing and processing requirements in fog computing systems, as shown in the third row of Table 2.

Table 2. The main requirements of a fog node and the related main operations

Main requirements	Main operations	Basic Infrastructure or algorithms	Examples
Interactivity	Sensor to sensor Sensor to edge (fog) Fog to cloud	Communication protocols Synchronization Rout detection	LAN, WLAN, Dijkstra
Computing	Data gathering Data processing Data sending	Normalization Compression Data/ signal filtering Cryptography	Sort, MPEG, DCT, FIR, SHA, CRC
Management	Data flow Fault tolerance Software-reconfiguration	Middle-ware Intelligent framework	OMAP (Open Multimedia Applications Platform), Microsoft Azure
Energy harvesting	–	Energy transferring filters	Omega, SFS (Sandia Frequency Shift)

To our knowledge, there is no standardized micro-architecture evaluation benchmark for the fog/edge computing domain. Therefore, we propose a benchmark based on commonly reported applications in the literature, as listed in Table 3 and discussed

in literature like [17]. While our proposed benchmark is tailored to a specific context, our contribution is widely applicable, as the NIMA methodology can be adapted to other benchmarks. Our findings, based on the collected benchmarks, support the suitability of our proposed utilization metric and its derivation method, as well as the additional parameters outlined in Table 7, which align with our design objectives. It should be noted that these metrics and parameters are not fixed and can be adjusted to suit the specific benchmark being considered.

Table 3. The main requirements of a fog node and the related main operations

Benchmark / FUs	ADPCM	AES	SHA	blowfish	CRC-32	GSM	JPEG	Motion	DCT	FFT	FIR	Qsort	Sort	Bdist	Dijkstra
#signed-add-32	19	4	5	2	1	16	16	7	3	7	2	3	2	1	2
#signed-sub-32	2	-	-	2	-	2	8	2	-	2	1	2	1	-	-
#signed mul-32	31	1	-	-	-	15	4	1	2	4	1	-	-	-	1
#signed-comp32	8	9	2	7	1	21	18	6	1	7	1	4	5	1	3
Bitwise-OR-32	1	-	4	2	-	1	2	2	-	-	-	1	-	-	-
Bitwise XOR32	-	12	2	4	1	1	1	2	-	-	-	-	-	-	-
Bitwise-AND32	-	-	2	-	-	2	1	-	-	-	-	-	-	-	-
Mem-Dualport	2	2	2	2	1	2	2	2	2	2	2	2	1	-	2
ShiftRa-32	2 , 2	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Shift-ll32	-	-	-	-	-	9	1	1	-	-	-	-	-	-	-
Shift-rl32	-	-	-	-	-	1	1	1	-	-	-	-	-	-	-

5.2. Fog Data-path Element Extraction

To obtain optimal data-path features, such as component type and usage statistics, for a fog computing benchmark, we leverage the internal passes of a typical open source HLS flow. Our HLS tool of choice is the LegUp high-level synthesis framework v4.0 [36], which compiles a C function or program to full hardware. Information is extracted from the LLVM passes corresponding to allocation and binding steps, prior to final RTL code generation, as explained in Section 4.2. Tool-specific optimizations are employed for each benchmark algorithm to achieve the optimal data-path. Using the control-data flow graph (CDFG) and corresponding basic blocks in scheduling mode, we extract the necessary number of data-path modules and their simultaneity in different scheduled cycles, as summarized in Table 3.

Table 4. The normalized simultaneity distribution of the compare module in the algorithms of the benchmark

Conc. level(l)	$\mu_{comp,f}(l)$														$\mu_{u,m}(l)$	$\mu_{lb,comp}(l)$
	ADPCM	AES	blowfish	GSM	MPEG	SHA	FFT	FIR	DCT	sort	CRC32	Bidist	Qsort	Dijkstra		
1	0.16	0.287	0.14504	0.37906	0.416	0.24	0.055	0.192	0.121	0.267	0.143	0.25	0.226	0.1682	0.2178	0.21780
2	0.08	0.076	0.01527	0.04332	0.04	0.06	0.006	0	0	0	0	0	0	0.0093	0.0236	0.04715
3	0	0.029	0	0.00361	0.016	0.02	0	0	0	0	0	0	0	0	0.0049	0.01475
4	0	0.012	0	0.00361	0	0	0	0	0	0	0	0	0	0	0.0011	0.00437
5	0	0.006	0	0.00361	0	0	0	0	0	0	0	0	0	0	0.0007	0.00337
6	0	0	0	0.00361	0	0	0	0	0	0	0	0	0	0	0.0003	0.00154
7	0	0	0	0.00361	0	0	0	0	0	0	0	0	0	0	0.0003	0.00180
8	0	0	0	0.00361	0	0	0	0	0	0	0	0	0	0	0.0003	0.00206

Slight differences in data-path elements between the NISC architecture library and LegUp basic operations necessitate conversion of some benchmark algorithm C code to extract data-path features. The most important one, is division. If a hardware division unit is missing from a simple NISC data-path, a software solution is used due to compiler limitations. Moreover, when more efficient, division of a variable with a small fixed number is replaced with a shift operator.

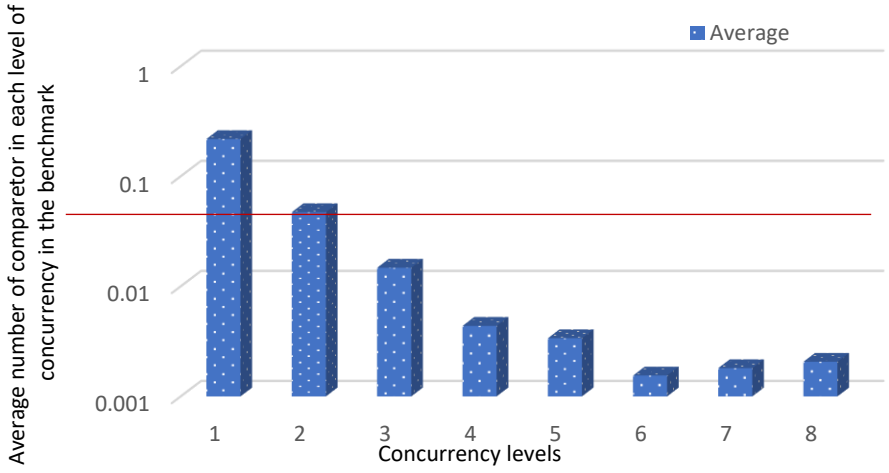

Figure 3. Histogram of usage statistics of compare FU in the suggested benchmark with specifying $\mu_{lb,FU}$ (The red line)

In Section 4.2, we discuss how the extracted module information is analyzed to obtain usage statistics for FUs in the benchmarks. To better understand the parameters' interrelation, we present the effect of adjusting the concurrency level knob for the comparator data-path element in Table 4. The table shows the normalized abundance of the comparator module in the scheduled cycles of each benchmark algorithm, providing a vertical view of the module's simultaneity distribution throughout the scheduling cycles. Normalization is based on the total scheduling cycles of each algorithm, as per Equation 1 ($\mu_{comp,f}(l)$ in Table 4). The average number of each concurrency level is then calculated using Equation 2 for a benchmark (column $\mu_{u,M}(l)$). Finally, we construct associated FU histograms based on each level of FU concurrency in the benchmark cycles ($\mu_{u,M}(l)$) in Figure 3.

In this stage, the designer determines the exact number of each FU on the corresponding histogram by choosing μ_{lb}. As a rule of thumb, different concurrency levels $CL = \{1_1, \dots, l_L\}$ we suggest an initial value for this parameter as a *utilization lower bound function* $\mu_{lb,FU}: N^+ \to R$.

$$\mu_{lb,FU} = \frac{\sum_{l=1}^{L}(l.\mu_{u,M}(l))}{\sum_{f=1}^{F}\sum_{l=1}^{L}(\mu_{u,f}(l))} \qquad (4)$$

Equation 4, computes the weighted average concurrency level of individual functional units in benchmark algorithms.

It does so by multiplying the average number of each FU concurrency level in benchmark with each level of concurrency in the numerator. This value is then divided by the sum of utilization functions related to the different concurrency levels in the denominator. This approach takes into account the impact of varying clock cycles on the benchmark, in order to equalize the effect of utilization across all cycles. It also accounts for the effect of different concurrency levels on functional unit utilization. As mentioned in Section 4.1, separate utilization lower bounds are calculated for each FU based on its usage statistics. Table 5 summarizes the exact values of $\mu_{lb,FU}$ and the corresponding value of each FU ($n_{u,M,\mu_{lb}}$) for the proposed benchmark. For example, the histogram chart in Figure 3 shows the calculated value of the utilization limitation (μ_{lb}) for the comparator, represented by a red line. Note that the chart depicts the number of FUs covered by altering the lower bound parameter based on design objectives. It is worth noting that this parameter can also be specified using the minimum, maximum, or average of the FU distribution in different concurrency levels; however, experiments suggest that focusing on the abundance of FU concurrency levels is necessary to determine the optimal number of each FU for the proposed data-path.

Table 5. The exact number of FUs regard to the calculated μ_{lb} for the fog computing benchmark

FUs	$\mu_{lb,FU}$	$n_{u,M,\mu_{lb}}$
Adder	0.01332227	4
Mul	0.02982095	2
Sub	0.09274926	2
Comp	0.08406878	1

Table 6. Effect of different RF modifications on the performance and cost metrics

Processors	NMIPS	1% utilization-based processor					
RF conf.	RF3x2	RF8x4	RF8x4 IR	RF8x4 OR	RF4x2	RF4x2 IR	RF4x2 OR
power (mW)	71.42	69.17	53.87	73.28	57.15	65.78	63.32
Delay (ns)	17.62	16.012	10.83	15.24	16.005	10.69	16.005
Area	5981.66	7633.66	7167.67	7500	6132.67	5982.33	6886.33

5.3. The Data-path Design Flow

We propose values for utilization-based parameters in the design flow for each design objective, as listed in Table 7. To motivate these parameters, we studied various sizes of RF ports, pipeline registers, and forwarding links on data-paths with module numbers based on a one percent utilization lower bound. The maximum required concurrency level for this utilization lower bound is four. We also implemented NMIPS processor with RF3x2 for comparison, which is a synthesizable hardware description generated for NISC-style MIPS [27]. Table 6 summarizes the average performance and cost parameters related to the synthesis of benchmark algorithms on the designed architectures and NMIPS. In Table 6, the input and output registers of each functional unit are denoted as IR and OR, respectively. Moreover, forwarding links are enabled in all data-paths under study.

Table 7. The proposed design parameters for various design objectives

Objective	RF input port size	Pipeline registers
Performance	$RF_d = \max(\max(n_{u,M,lb}), \gamma_{tec})$	$R_d = \sum(FU)_i$, $(i = 1, \ldots, n)$
Area	$RF_a = \max(\max(n_{u,M,lb})/2, \gamma_{tec}/2)$	$R_a = \sum(n_{u,M,lb})_i$, $(i = 1, \ldots, n)$
Power	$RF_p = \max(\max(n_{u,M,lb})/2, \gamma_{tec}/2)$	$R_p = \sum(n_{u,M,lb})_i$, $(i = 1, \ldots, n)$

The synthesis results indicate that the performance and cost metrics are impacted by the size of RF ports, pipeline registers, and forwarding links. For optimal performance, RF port size (RF_d) can be adjusted based on the maximum level of FU simultaneity ($n_{u,M,\mu_{lb}}$). However, exceeding a certain threshold for RF port size can lead to negative outcomes due to the complexity of the data-path. Therefore, the system architect should consider the parameter representing technology limitation (γ_{tec}) to avoid this. Table 6 shows that increasing RF port size and pipeline registers in the data-path can enhance performance and power consumption but can result in increased area. To balance performance and power consumption, a moderate RF port size (RF4x2) should be maintained, and more simultaneity should be achieved using pipeline registers and forwarding techniques. Also, if the RF port size is maximum, input registers further reduce power consumption, while in a data-path with a moderate or smaller RF port size (RF4x2), output registers are more useful for reducing power. Forwarding links should be enabled in all data-path designs, particularly when the RF port size is smaller than the required number of concurrency. The maximum RF port size that a NISC tool-set can handle is RF16x8, which is γ_{tec} in our design. Table 8 shows the exact values of design parameters for the suggested benchmark. These values must be established on the proposed data-path architecture shown in Figure 5.

Table 8. The proposed base processor architecture

Objective	Performance	Area	Power
FU Numbers	Adder Mul Sub Comp 4 2 2 1	Adder Mul Sub Comp 4 2 2 1	Adder Mul Sub Comp 4 2 2 1
RF input-port	4	2	2
Pipeline Reg	IReg, 20	IReg, 14 / OReg, 7	OReg, 7
Forwarding	1	1	1

5.4. Performance, Area, and Power Objectives

The utilization lower bound (μ_{lb}) acts as a tuning mechanism to balance processor performance and area. The utilization histogram illustrates the trade-off between different μ_{lb} values. Decreasing μ_{lb} increases the number of functional units and boosts performance, but it also leads to greater area consumption. Conversely, raising μ_{lb} reduces the number of functional units, resulting in less area consumption but potentially lower performance (see Figure 4). Power consumption is influenced by both performance and area usage. Nevertheless, we develop a general approach to evaluate the impact of μ_{lb} on power variations. To investigate the effect of μ_{lb} on power consumption, we initially create corresponding architectures without any adjustments. Our experimental results demonstrate that reducing μ_{lb} leads to an increase in power consumption. This increase stems from the increased complexity of the data-path due to the use of additional functional units. Therefore, the utilization parameter should be considered along with other important factors such as data-path pipelining and forwarding [29], to determine the optimal power criteria. By analyzing power consumption at different lower bounds through our design algorithm, we can assess the impact of design parameters on power usage and determine the power optimum architecture. Notably, we observed a decrease in power consumption at the 1% point in Figure 4, indicating that our proposed solutions can effectively manage data-path complexities and balance performance and power trade-offs.

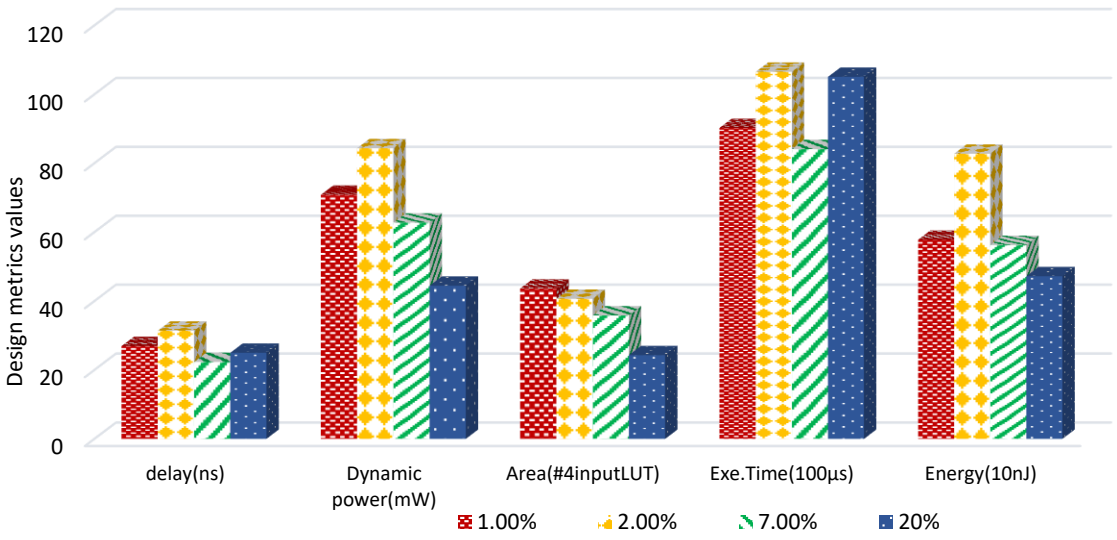

Figure 4. The impact of proposed design parameters on design metrics for different utilization lower bounds

5.5. *The NISC Tool-set Flow*

The extracted parameters from the preceding section help with designing the required data-path for the desired performance and cost metrics. Then, the data-path and the C codes of the applications are inserted into the NISC flow to achieve the final system.

5.5.1. *The Proposed Architecture Features*

The designed data-path is integrated through a parametric design flow, with a focus on the concurrency of functional units (FUs). The resulting architecture capitalizes on modified input/output RF port sizes, along with pipeline FUs and forwarding paths, to achieve the required data-path parallelism, as shown in Figure 5. Large RF buses are replaced by small multiplexers in the resulting data-path. The design algorithm determines the presence of pipeline registers and forwarding paths based on the desired objectives.

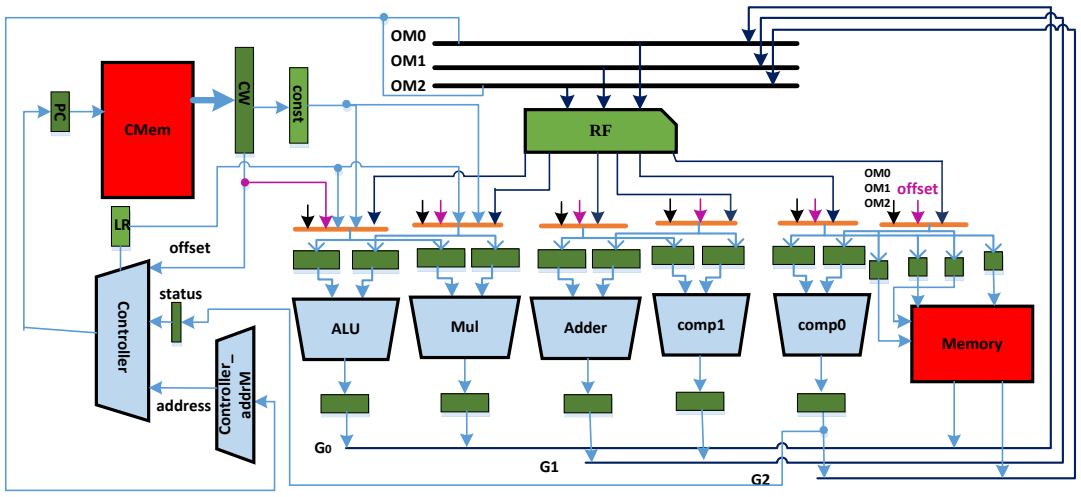

Figure 5. The proposed base processor architecture

The suggested architecture model employs a hierarchical view of the data-paths, dividing functional units (FUs) into categories based on the maximum number of concurrent FUs ($\max(n_{u,M,\mu_{lb}})$). Each category contains identical FUs that can be executed separately for the required simultaneity, with FUs allocated to each group in a round-robin style. The output of each group is connected to an RF input port via a multiplexer as a bus module (i.e., the OM modules in Figure 5). To establish forwarding paths, the FU output port is connected to all FU input ports via RF output and input multiplexers. Customization of this architecture template is achieved by determining the required data-path modules and connections in the design flow, based on specific design goals. Note that this proposed architecture is customized from the MIPS processor.

5.6. *Simulation Results and Discussion*

After finalizing the desired data-path based on performance or cost objectives, three fog processing architectures are produced and benchmark algorithms are implemented on each. The optimum architecture is compared with the NMIPS architecture available in the NISC library. The RTL design description code automatically produces the architecture model in Verilog language after initialization to the NISC tool-set, and is reported with two specific models: a general model and a Xilinx model. Modelsim 6.5 is used for simulation with the general model, while the ISE tool-set synthesizes the Xilinx model for required evaluations. The RTL description generated for the synthesis flow is in accordance with the XC4VLX60 device linked to the FPGA platform of Virtex4.0 and Virtex6.0, with a related package and speed of ff896 and -6, respectively [30].

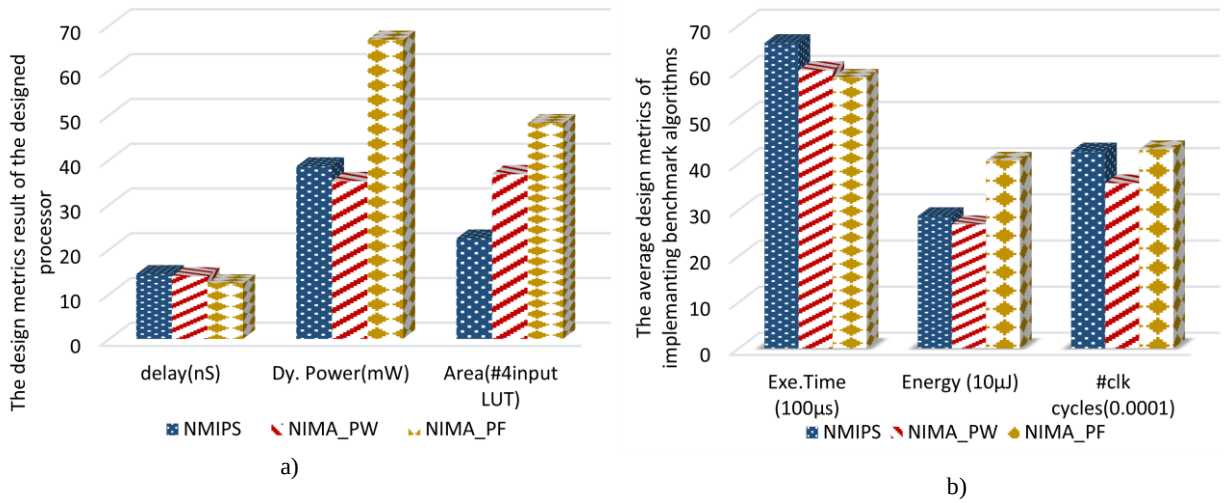

Figure 6. Comparison of the base NMIPS with the benchmark-driven NIMA architecture: a) The synthesized processors' metrics, b) The benchmark execution results

Figure 6 displays the results of implementing benchmark algorithms on the proposed architectures and the NMIPS processor, with NIMA_PW and NIMA_PF representing optimum power and performance architectures. Comparisons are made based on five measures: performance, power, area, execution time, and energy consumption. As discussed in Section 5.3, an architecture with optimum area is almost identical to one with optimum power, with the only difference being the location of pipeline registers. Therefore, we consider the area of the NIMA_PW as the area related to an architecture with optimum area.

The simulation results demonstrate improved performance in the NIMA_PF architecture. This processor is 14.5\% in delay and 12% in execution time better than NMIPS. Note that, the increment of FUs and other modules like pipeline registers complicate the proposed architectures than NMIPS besides improving the latency. This latency improvement inevitably increases power consumption as well. Although the proposed procedures provide for an acceptable decrease in power in NIMA_PW than the NMIPS for the intended benchmark. The power consumption of NIMA_PW is 9.2% lower than that of NMIPS, resulting in 6.6% less energy consumption. This improvement is attributed to the optimized size of RF, register file, and pipeline registers, coupled with appropriate placement of these components and forwarding links, leading to a favorable trade-off between power and performance. Between the proposed designs, NIMA_PW consumes 30.9% less area than NIMA_PF. It is due to the reduced size of RF ports and fewer pipeline registers.

6. Conclusions

This article introduces NIMA, a design methodology that leverages the NISC paradigm to create domain-specific (micro-) architectures. NIMA utilizes a parametric design flow to accelerate the design process based on specific objectives and a proposed utilization metric, rather than exhaustively exploring the design space. Parameters for the design flow are determined from the utilization metric, which measures the abundance distribution of modules in various clock cycles of scheduled benchmark algorithms, with the utilization lower bound serving as a measurement criterion for the designer. The proposed algorithm determines the optimal number of FUs, size of RF ports, number and position of pipeline registers, and forwarding links based on the design objectives. In light of the challenges posed by fog embedded systems, the resulting customized processor architecture, produced through the NIMA design flow, offers improved power and performance compared to the basic processor, validating the efficacy of the design flow and proposed design parameters. This systematic design flow can be applied to optimize any processor family based on desired design objectives, with the added advantage of reduced time-to-market due to the ease of changing the data-path according to specified objectives.

7. REFERENCES

[1] Gubbi J, Buyya R, Marusic S, Palaniswami M. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 2013, 1645-1660.
[2] Zhang S, Zhang F, Wu Y, He B, Johns P. Hardware-conscious stream processing: A survey, *SIGMOD Rec.*, 2020, 18-29.
[3] Adegbija T, Rogacs A, Patel C, Gordon-Ross A. Microprocessor optimizations for the internet of things: A survey, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018, 7-20.
[4] Sun N, Xiang and Ansari. EdgeIoT: Mobile Edge Computing for the Internet of Things, *IEEE Communications Magazine*, 2016 22-29.
[5] Bonomi F, Milito R, Zhu J, Addepalli S. Fog computing and its role in the internet of things, Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, *ACM*, 2012, pp. 13-16.
[6] Das R and Inuwa M M. A review on fog computing: Issues, characteristics, challenges, and potential applications, *Telematics and Informatics Reports*, 2023, 100049.
[7] Morabito R, Beijar N. Enabling Data Processing at the Network Edge through Lightweight Virtualization Technologies, IEEE International Conference on Sensing, Communication and Networking, 2016, no. 607728.
[8] Shi W, Cao J, Zhang Q, Li Y, Xu L. Edge Computing: Vision and Challenges, *IEEE Internet of Things Journal*, 2016, 637-646.

[9] Rizk M, Baghdadi A, Jezequel M, Mohanna Y, Atat Y. NISC-Based Soft-Input-Soft-Output Demapper, *IEEE Transactions on Circuits and Systems II*, 2015, 1098-1102.
[10] {More}More P. Review of implementing fog computing, *International Journal of Research in Engineering and Technology*, 2015, 2319-2322.
[11] Gautschi M, Member S, Schiavone P D, Member S, Traber A, Loi I, Pullini S, Rossi D, Flamand E, Frank K G. A near-threshold riscv core with dsp extensions for scalable iot endpoint devices, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2016, 2700-2713.
[12] Wang Z, Liu Y, Zhang D. An energy-efficient heterogeneous dual-core processor for internet of things, *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2015, pp. 2301-2304.
[13] Yuxiang H, Ning M, Stefan B, Zou Z, Lirong Z. A 61 ua/mhz reconfigurable application-specific processor and system-on-chip for internet-ofthings, *28th IEEE International System-on-Chip Conference (SOCC)*, 2015.
[14] Woo Oh H., Lee S. E. The Design of Optimized RISC Processor for Edge Artificial Intelligence Based on Custom Instruction Set Extension, *IEEE Access*, 2023, pp.49409 - 49421.
[15] Chen Y, Lu S, Kim H S, Blaauw D, Dreslinski R G, Mudge T. A Low Power Software-Defined-Radio Baseband Processor for the Internet of Things, *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016, pp. 40-51.
[16] Liu Z, Jiang J, Lei G, Chen K, Qin B and Zhao X. A Heterogeneous Processor Design for CNN-Based AI Applications on IoT Devices, *Procedia Computer Science*, 2020, pp. 2-8.
[17] Johann S, Moreira M, L Heck, Calazans N, Hessel F. A processor for IoT applications: An assessment of design space and trade-offs, *Microprocessors and Microsystems*, 2016, 156-164.
[18] Sisto A, Pilato L, Serventi R, Saponara S, Fanucci L. Application specific instruction set processor for sensor conditioning in automotive applications, *Microprocessors and Microsystems*, 2016, 375-384.
[19] Rizk M, Baghdadi A, Jezequel M, Mohanna Y, Atat Y. Design and prototyping flow of NISC-based flexible MIMO turbo-equalizer, *IEEE International Symposium on Rapid System Prototyping*, 2014, pp.16-21.
[20] Rizk M, Baghdadi A, Jezequel M, Mohanna Y, Atat Y. Design and Prototyping Flow of Flexible and Efficient NISC-Based Architectures for MIMO Turbo Equalization and Demapping, *Electronics Journal*, 2016, 50.
[21] Rizk M, Baghdadi A, Jezequel M, Mohanna Y, Atat Y. No-instruction set-computer design experience of flexible and efficient architectures for digital communication applications: two case studies on MIMO turbo detection and universal turbo demapping, *Design Automation for Embedded System*, 2021, 1-42.
[22] Rizk M, Baghdadi A, Jézéquel M, Y Atat, Mohanna Y. NISC-Based MIMO MMSE Detector, *Journal of Circuits, Systems and Computers*, 2150069, 2021.
[23] Zeng D, Gu L, Guo S, Cheng Z, Yu S. Joint Optimization of Task Scheduling and Image Placement in Fog Computing Supported Software-Defined Embedded System, *IEEE Transactions on Computers*, 2016, 3702-3712.
[24] Brzoza-Woch R, Konieczny M, Nawrocki P, Szydlo T, Zielinski K. Embedded systems in the application of fog computing-Levee monitoring use case, *IEEE International Symposium on Industrial Embedded Systems*, 2016.
[25] Gorjiara B, Reshadi M, Gajski D. Low-power design with nisc technology, *InLecture Notes in Designing Embedded Processors*, Henkel S, Jorg ang Parameswaran, Springer, 2007, Ch. 2.
[26] Mishra S K, Rajawat A, Singh A P, Application Oriented Analysis of NISC Architecture, *International Journal of Engineering Science and Technology*, 2010, 3412-3421.
[27] Gorjiara B, Gajski D. Custom processor design using NISC: A case-study on DCT algorithm, *3rd Workshop on Embedded Systems for Real-Time Multimedia*, 2005, pp. 55-60.
[28] Reshadi M, Gorjiara B, Gajski D. Utilizing horizontal and vertical parallelism with a no-instruction-set compiler for custom datapaths, *International Conference on Computer Design: VLSI in Computers and Processors*, 2005, pp. 69-74.
[29] Rabaey J. Low Power Design Essentials, *Springer*, New York, USA, 2009.
[30] Maabi S, Attarzadeh-Niaki S H, Abbaspour M. OMID: Optimized MIcro-Architecture Design for Fog Computing Applications, *27the Iranian Conference on Electrical Engineering (ICEE)*, IEEE, 2019, pp. 2033-2038.
[31] Lee W, Nam K, Roh H g, Kim S h. A Gateway based Fog Computing Architecture for Wireless Sensors and Actuator Networks, *8th International Conference on Advanced Communication Technology (ICACT)*, 2016, pp. 210-213.
[32] Kraemer F A, Braten A E, Tamkittikhun N, Palma D. Fog Computing in Healthcare-A Review and Discussion, *IEEE Access*, Vo. 5, 2017, 9206-9222.
[33] Rahmani A M, Gia T N, Negash B, Anzanpour A, Azimi I, Jiang M, Liljeberg P. Exploiting smart e-health gateways at the edge of healthcare internet-of-things: A fog computing approach, *Future Generation Computer Systems*, 2018, 641-658.
[34] Thien A T, Ricardo C -P. A Systematic Literature Review of Fog Computing, {\it NOKOBIT}, 2016, 28-30.
[35] Ilija R, Ivan P, Dejan D. Multi channel sensor measurements in fog computing architecture, *Zooming Innovation in Consumer Electronics International Conference (ZINC), IEEE*, 2017, 9-12.
[36] Canis A, Choi J, Aldham M, Zhang V, Kammoona A, Czajkowski T, Brown S D, Anderson J H. Legup: An open-source high-level synthesis tool for fpga-based processor/accelerator systems, *ACM Trans. Embed. Comput. Syst*., 2013, 24:1-24:27.

Somayeh Maabi received the B.S. and M.S. degrees in computer architecture engineering from Islamic Azad university of Qazvin branch, in 2008 and 2013 respectively and she is the PHD candidate in computer architecture in Shahid Beheshti University, Tehran, from 2013. Her current research interests include embedded system design, ASIP and ASIC design especially in fog computing domain.

Seyed-Hosein Attarzadeh-Niaki is an assistant professor in computer systems architecture at Shahid Beheshti University GC (SBU), Tehran, Iran. His research interests include modeling and design methodologies for smart and safe realtime embedded and cyber-physical systems.

Maghsoud Abbaspour received the B.S., M.S., and Ph.D. degrees in electrical engineering from the University of Tehran, Tehran, Iran, in 1992, 1995 and 2003, respectively. He is an associate professor of Faculty of Computer Science and Engineering and director of Computer Networking and Network Security Laboratory in Shahid Beheshti University, Tehran, Iran since 2005. He is interested in Wireless Sensor Networks, peer to peer and ad hoc networks, network security and Internet of Things.