Microprocessors and Microsystems 106 (2024) 105040

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

A light-weight neuromorphic controlling clock gating based multi-core cryptography platform

Pham-Khoi Dong [a], Khanh N. Dang [b], Duy-Anh Nguyen [a], Xuan-Tu Tran [a,*]

[a] *VNU Information Technology Institute, Vietnam National University, Hanoi (VNU), Hanoi 123106, Vietnam*
[b] *Adaptive Systems Laboratory, Graduate School of Computer Science and Engineering, The University of Aizu, Aizu-Wakamatsu, Fukushima, 965-8580, Japan*

A R T I C L E I N F O

Keywords:
AES
High-throughput
Multi-core
Cryptography
Spiking neural network
Neuromorphic
Brain-inspired computing

A B S T R A C T

While speeding up cryptography tasks can be accomplished by using a multi-core architecture to parallelize computation, one of the major challenges is optimizing power consumption. In principle, depending on the computation workload, individual cores can be turned off to save power during operation. However, too few active cores may lead to computational bottlenecks. In this work, we propose a novel platform named Spike-MCryptCores: a low-power multi-core AES platform with a neuromorphic controller. The proposed Spike-MCryptCores platform is composed of multiple AES cores, each core is equipped with a clock-gating scheme for reducing its power consumption while being idle. To optimize the power consumption of the whole platform, we use a neuromorphic controller. Therefore, a comprehensive framework to generate a data set, train the neural network, and produce hardware configuration for the Spiking Neural Network (SNN), a brain-inspired computing paradigm, is also presented in this paper. Moreover, Spike-MCryptCores integrates the hardware SNN inside its architecture to support low-cost and low-latency adaptations. The results show that implemented SNN controller occupies only 2.3 % of the overall area cost while providing the ability to reduce power consumption significantly. The lightweight SNN controller model is trained and tested with up to 95 % accuracy. The maximum difference between the predicted number of cores and the ideal one from the label is one unit only. Under 24 test scenarios, a SNN controller with clock-gating helps Spike-MCryptCores reducing the power consumption by 48.6 % on the average; by 67 % for the best-case scenario, and by 39 % for the worst-case scenario.

1. Introduction

Cryptography has been used for thousands of years to hide secret messages, with the first known evidence found in an inscription carved around 1900 BC in Egypt. Evidence of cryptography can be seen in major early civilizations, such as "Arthshashtra" in India. Julius Caesar used a substitution cipher to convey secret messages to his army generals, known as the Caesar cipher. In the 16th century, Vigenere designed a cipher that used an encryption key, which could be broken by using the frequency of letters in the language. Hebern designed an electro-mechanical contraption called the Hebern rotor machine in the 19th century, which used a single rotor embedded in a rotating disc to encode a substitution table. The Engima machine was invented by German engineer Arthur Scherbius at the end of World War I and was heavily used by German forces during the Second World War. The Enigma machine used multiple or more rotors, with the key being the initial setting of the rotors. Post-World War II, cryptography attracted commercial attention, with businesses trying to secure their data from competitors. IBM formed a "crypto group" headed by Horst-Feistel in the 1970s and designed a cipher called Lucifer. In 1973, the Nation Bureau of Standards (now NIST) in the US requested proposals for a block cipher, which was accepted and named DES or the Data Encryption Standard. However, DES was broken by an exhaustive search attack in 1997 due to its small size.

The Advanced Encryption Standard (AES) was developed by Belgian cryptographers, *Vincent Rijmen* and *Joan Daemen*, and later published by the National Institute of Standards and Technology (NIST) in 2001 [1]. The AES is now widely regarded as the most popular symmetric cryptosystem with a huge variety of applications.

With the emergence of distributed computing [2,3] and the rise of big data [4], the need for secure and rapid data transmission has become ubiquitous [5]. Encryption is a technique that manipulates image data to prevent illegal access. Over the past two decades, various encryption algorithms have been developed for digital image security. These

* Corresponding author.
E-mail address: tutx@vnu.edu.vn (X.-T. Tran).

https://doi.org/10.1016/j.micpro.2024.105040
Received 16 September 2023; Received in revised form 27 November 2023; Accepted 22 February 2024
Available online 26 February 2024

schemes are divided into full and partial encryption. Full encryption uses a lot of resources and time, while partial encryption is computationally efficient and suitable for real-time applications like teleconferencing and camera surveillance. In recent years, chaotic maps have gained attention in various fields, including mathematics, physics, computer science, and engineering due to their sensitivity, blanketing nature, and topological transitivity [6–8]. propose a DNA key-based visual image encryption scheme using quantum chaotic map. Initial conditions are computed from a DNA sequence, plaintext image, and SHA-512 hash function. Three random vectors are generated, with two used for correlation breaking and the third used in XORed operation. The scheme is designed to resist cryptographic attacks by transforming the ciphertext into a visually encrypted image. The diffused image is divided into Least Significant Bit (LSBs) and Most Significant Bits (MSBs), and more DWT is applied to the carrier image. Experimental analyses confirmed the scheme's ability to withstand attacks and that it is noise-resistant.

To ensure data security, it is imperative to protect the data using a recognized security standard such as AES. Furthermore, large-scale systems also demand high bandwidth and low latency capabilities to process data in real-time. Consequently, the design of hardware encryption modules and their integration into System-on-Chips (SoCs) as accelerators has gained significant traction. Researchers have explored various approaches to enhance the throughput of AES encryption and decryption processes while designing AES hardware accelerators [9–13].

Since the AES computation is performed separately for every 128 bits of data, one natural approach to accelerate the process is to replicate the AES core into multiple instances and perform the encoding parallelly [14]. While this approach can indeed enhance the throughput, it does come with several drawbacks:

- First, if the incoming data fully utilizes all the parallel cores, we can benefit from the increased throughput. However, if incoming data is sparse, such as only utilizing 30% of the cores, there are some operating cores without any load (i.e., 'no-load' cores), resulting in unnecessary power consumption without performing any computations.
- Secondly, the power of 'no-load' cores can be significantly saved by implementing clock-gating or power-gating techniques. However, the process of turning 'no-load' cores on and off leads to many challenges as it requires a certain number of clock cycles for readiness. Consequently, the system should adapt over a period of T cycles. However, the incoming data rate can vary, and the system must decide on an optimal number of cores being activated or deactivated. If too many cores are activated, unused cores will consume unnecessary power. Conversely, if too few cores are activated, the system will struggle to process all incoming data, resulting in bottlenecks in the data flow. Hence, the system must employ a dedicated controller to decide the number of cores to be turned on/off, and the efficiency of this controller is crucial for the overall performance of the system.
- Thirdly, conventional control algorithms such as Fuzzy Logic or PID can assist the system in adapting to specific scenarios. However, these algorithms face two challenges: (1) they demand significant resources, making it difficult to design dedicated hardware; and (2) extending these algorithms to accommodate new scenarios is not straightforward. Although Artificial Neural Network (ANN)-based solutions offer flexibility, the hardware complexity associated with ANN implementations can still present many issues. Alternatively, Spiking Neural Network (SNN) utilizing the Leaky-Integrate-and-Fire neuron model, a brain-inspired computing paradigm, can serve as a viable option for the controller. SNNs offer a lower area cost while delivering comparable performance. Due to their simplified computation, hardware implementation of SNNs can be integrated into our system with minimal area overhead. Moreover, as a neural network approach, SNNs can be trained and adapted to various scenarios.
- Fourthly, SNNs have demonstrated their efficacy in various domains, including robotics, control systems, and computer vision. However, to the best of our knowledge, no previous work has explored the adaptation of multiple AES cores using SNNs. The primary challenges in this context lie in data preparation, SNN training, and the implementation of SNNs into dedicated hardware modules.

In this work, we address the aforementioned challenges by introducing Spike-MCryptCores, a brain-inspired low-power multi-core AES platform. The Spike-MCryptCores platform utilizes multiple AES cores to enhance the total performance. To minimize power consumption, Spike-MCryptCores uses a clock-gating scheme for each individual core. To control the clock gating process effectively, Spike-MCryptCores employs a controller based on spiking neural networks (SNNs) inspired by the human brain. SNNs have garnered significant attention recently due to their relative simplicity and low-power hardware characteristics [15–18]. From a range of available neuron models, we have chosen the Leaky-Integrate-and-Fire model due to its ability to strike a balance between simplicity and bio-plausibility. The main contributions of this work are as follows:

- We introduce an open-source platform that enables dataset preparation, training, and testing of SNN (Spiking Neural Network) for clock-gating control in a multiple AES cores system[1]. Additionally, we propose a method for quantizing and generating hardware SNN configurations that allows the system to have a dedicated controller to adapt to different scenarios. The SNN achieves over 95 % accuracy using a lightweight controller that occupies only 2.3 % of the total system area cost.
- We develop a hardware architecture of the proposed a low-power multi-core AES platform that incorporates two significant innovations: (1) a tailored, cost-effective hardware architecture for SNN that supports both input current and spike input modes for system control, and (2) the integration of the SNN hardware into the multi-core AES platform to establish a low-power multi-core AES platform thanks to SNN-based clock-gating scheme. Consequently, by implementing this architecture, power consumption can be reduced by 39 % to 67 %, while the system incurs less than a 10 % increase in area overhead.

To the best of our knowledge, this is the first-ever work on adapting a brain-inspired computing model to reduce power consumption in a multi-core system.

The remaining part of this paper is organized as follows. Section 2 reviews related works on single-core and multi-core AES architectures. Section 3 presents our proposed parallelized multi-core architecture and how it operates. In Section 4, we provide in detail the implementation of the proposed architecture on a CMOS 45 nm technology and an evaluation of the SNN controller. Section 5 discusses some limitations of the work. Finally, Section 6 offers concluding remarks.

2. Related works

2.1. Advanced encryption standard implementations

The Advanced Encryption Standard (AES) algorithm was standardized by NIST in 2001. Since then, there have been many implementations of the AES in both software and hardware. Software AES implementations can be easily performed using a CPU. However, they are generally considered to be less secure, slower, and more power-

[1] Source code is available here: https://github.com/khanhdang/Spike-MCryptCores

consuming [12,13]. To address the aforementioned issues, hardware-based AES implementations offer viable solutions. For resource-constrained devices, the AES is often implemented with basic iterative, 8-bit, 16-bit, 32-bit, and 64-bit datapath width architectures with one or two S-Boxes to optimize area overhead and minimize power consumption [13,14,19,20]. The disadvantage of these architectures is low throughput because of the use of loops. For applications that require high bandwidth, AES hardware architectures are typically designed using full-parallel modes [9–11], unroll architectures [21], or pipeline architectures [22]. These architectures provide high performance but at a high area cost and high energy consumption.

One of the common approaches to enhance the performance of the AES implementation is to utilize multi-core AES [14,21,22]. Pammu et al. [14] take advantage of a Multi-core Processor (AMP-MP) to achieve high throughput while maintaining the security of an Advanced Encryption Standard based on Counter with Chaining Mode (AES-CCM). The proposed AMP-MP is implemented on a CMOS 65 nm processor with an 8-bit asynchronous 9-core architecture. The authors of [23] provide a novel parallelization technique for the Advanced Encryption Standard based on the Galois/Counter Mode (AES-GCM). The method permits the creation of scalable streaming cores capable of processing multiple individually keyed packets each clock cycle on broad segmented buses. Multi-FPGA systems are possible due to the architecture's lack of core-to-core communication requirements. Work in [24] describes an efficient design technique for implementing AES on reconfigurable hardware devices. The authors demonstrated how to overcome the FPGA's 100 Gbps speed limit by utilizing four AES cores and four binary field multipliers. Four pipeline steps have been placed within the multiplication in order to shorten the critical path of the GHASH operation. The final GCM design is 44-layer and delivers 119 Gbps on Xilinx Virtex-5 chips.

2.2. Low-power techniques

Many design techniques have been developed to reduce power consumption, such as clock gating, power gating, multi-threshold voltage CMOS cells, multi-VDD technique, dynamic voltage and frequency scaling (DVFS), etc. [25]. Clock gating is extremely beneficial for lowering the power and energy consumption of digital devices. The authors of [26] constructed and tested an energy recovery timed pipelined multiplier equipped with an inbuilt resonant clock generator that generates a sinusoidal clock. The results indicate a 70 % decrease in clock-tree power consumption and an overall power savings of 25–69 % when compared to a multiplier employing a standard square-wave clocking system and matching flip-flops (FF). Shmuel Wimer and colleagues [27] describe a unique technique dubbed Look-Ahead Clock Gating (LACG). This approach generates the clock enabling signals for each FF one cycle ahead of time using the current cycle data for the FFs. Not only it is likely to eliminate the bulk of redundant clock pulses, but it also avoids the AGFF and data-driven timing limitations. Because the LACGs are represented at the RTL level, the gating clock implementation is much simplified. Additionally, the authors recommend employing a single LACG for two FFs to save hardware overhead and power consumption. LACG was evaluated using a 22 nm technology process. The testing findings indicate that this strategy saves 22.6 % of the clock power and 12.5 % of the system's total power usage.

Although there are numerous techniques for lowering power consumption, this work adopts clock-gating as the technique. Since the Spike-MCryptCores utilize multiple AES cores, turning off the clock signal of the unused core will help reduce the power consumption. Obviously, other techniques can be integrated into our platform as later discussed in Section 5.

2.3. Brain-inspired controlling methods

Brain-inspired computing approaches [28] have shown enormous promise and success in in diverse of applications, such as image classification, natural language processing, and self-driving systems. Many existing works use a variety of these approaches to control system resources such as performance, energy consumption, and temperature.

Power consumption is one of the most challenging issues in the design of multi-core systems. In [29], the authors used reinforcement learning to control DVFS and the operation states. Experiments demonstrate that the system improves energy efficiency. The authors of [30] offered a new application scheduling and DVFS solution named CARTAD that is based on reinforcement learning and is designed to decrease system temperature while ensuring application latency. Jung and Pedram [31] provided a power management framework for multiprocessor systems based on supervised learning that assesses the system's performance based on input characteristics and calculates the best voltage–frequency setting using a precomputed policy table. In [32], the authors presented a strategy based on online learning that utilizes different experts' processor clock frequency levels, which are then picked at runtime based on power consumption and performance penalty.

Moreover, Spiking Neural Networks (SNNs), which are considered the third and most advanced neural network architecture, are directly inspired by the operation of biological brains. SNNs can be implanted in both hardware and software. However, thanks to the low complexity of the Leaky-Integrate-and-Fire neuron model and synapses, SNN hardware has been a promising solution for power-efficient computing. Both academics and industry have been investing in how to implement SNNs in hardware architectures [33–39]. Unlike other techniques that require high resources, SNNs are lightweight and can be integrated into the controller like Spike-MCryptCores. SNNs have been applied to multiple control applications such as robotics or decision making. In [40], the authors use TrueNorth neuromorphic chip by IBM to perform a closed-loop control. Path planning algorithm is also implemented in [41] using spiking neural networks. Fischl et al. [42] have demonstrated a self-driving robot using SNN.

Although SNNs (or neuromorphic computing) have shown their potential in several applications [36,40,41]. Applying SNNs for practical applications is still challenging. In [32], the authors went through deep analyses to compare conventional neural networks with SNNs.

Despite applying SNNs for controlling and computer vision tasks have been investigated, there is no work on how we can apply SNNs to control on-chip systems. In this work, we will try to answer the question of how we can train and implement lightweight SNNs to handle a controlling task within SoCs.

3. Spike-MCryptCores platform

In this section, we present the overview architecture of the proposed low-power multi-core AES platform with a brain-inspired controller (Spike-MCryptCores platform). Then, we will describe each module of this platform in detail.

3.1. Platform overview

The overview diagram of the proposed low-power multi-core AES platform (Spike-MCryptCores platform) is presented in Fig. 1.

In principle, the Spike-MCryptCores hardware architecture consists of N homogeneous AES cores operating in parallel. Due to the inconsistency of incoming data rates, there are chances that the incoming data will not fully utilize all N cores. Because there are cores not being used, Spike-MCryptCores will gate the clock to reduce the dynamic power consumption. However, the process of turning on/off each core has some delay and may mess up the order of the data. Therefore, we need to provide a proper controller to efficiently manage the multi-core system. In the Spike-MCryptCores platform, we use Spiking Neural Network (SNN), a brain-inspired computing model, as the backbone of the controller.

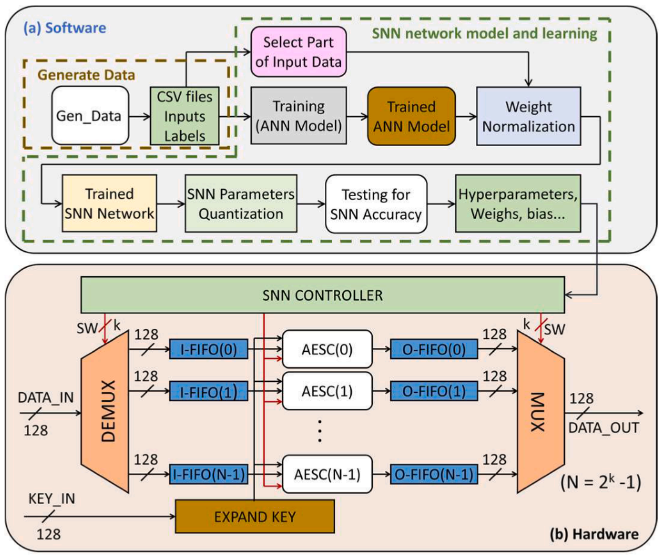


Fig. 1. Spike-MCryptCores platform (a) Software and (b) Hardware.

3.1.1. *Brain-inspired controlling*

As the incoming data can be varied at different rates at different times, we should apply a controlling mechanism to allow the system to enable/disable core during operation. Brain-inspired methods have been applied in robotic fields such as computer vision or decision making [28]. Inspired by the efficiency and the ability to adapt to multiple scenarios, the Spike-MCryptCores platform utilizes spiking neural networks, a brain-inspired computing paradigm, as the backbone. SNN computation is based on binary input (spikes/action potentials) and has shown the ability to control complex tasks with energy efficiency.

Fig. 2 illustrates the computing model of SNNs. The structure of a biological neuron is depicted in Fig. 2(a) where a neuron is connected to others via an axon-synapse-dendrite connection. Once a neuron issues an action potential (or spike), the spike is transmitted through the axon and goes to downstream neurons through different synapses. The incoming spike is received at a synapse and will be delivered to the core of a neuron (soma) via dendrite. The strength of a connection between

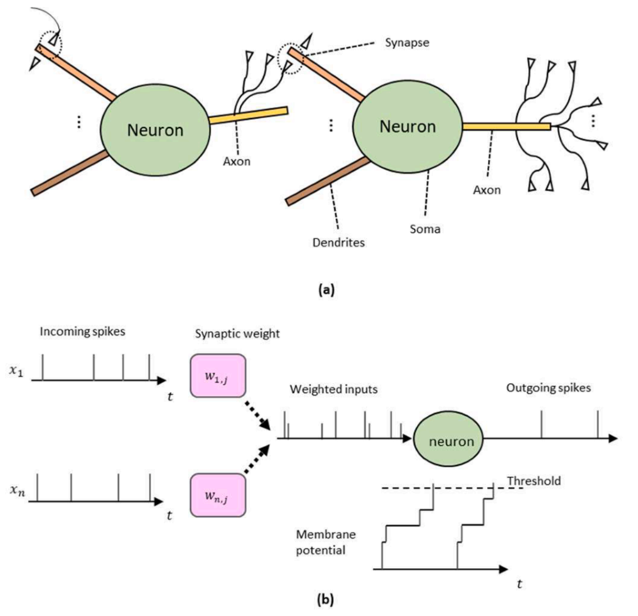


Fig. 2. Spiking neural network computing model: a) Biological neuron; (b) Spiking neuron.

neurons is usually considered as a weight in a conventional neural network. Fig. 2(b) shows the spike graphs for an SNN neuron, incoming spikes, usually considered as a binary value (current in biological neuron) goes through a synapse with a synaptic weight that alters the spike with different strengths. The weighted input will be integrated into the neuron's membrane potential. Once the membrane potential of a neuron crosses the threshold, the neuron issues a spike and sends it to the downstream neurons.

There are two basic ways of training an SNN model. The first method is to train with an ANN model and then convert it to an SNN [43]. This method provides the flexibility to allow designers to adapt to different scenarios. The second method is to train SNN with bio-inspired learning mechanisms [44,45] (STDP, SDSP) or ANN-like learning mechanisms (i. e., backpropagation) [46]. Both methods have demonstrated that SNNs can learn complex tasks. Furthermore, as a brain-inspired computing paradigm, using SNN can offer the possibility to adopt bio-plausible features.

3.1.2. *Software and hardware parts*

As illustrated in Fig. 1, the proposed platform comprises two major components: software and hardware. Fig. 1(a) describes the process performed on the software, including the following stages: (1) generate training data based on different scenarios of input data and save it to comma-separated values (.csv) files; (2) train the SNN model using the data scenarios generated in Stage 1, evaluate the model, and generate hyperparameters, weights, and biases for the Spike-MCryptCores hardware. The main role of the *Software* is to generate several common scenarios of data and use them to train the SNN. With the trained SNN model, the system can predict the behavior of the data rate and give a suitable adaptation. In other words, the SNN model will predict the number of AES cores needed to be turned on to encode the incoming data. If too many cores are activated, there are unused cores and it will waste power. On the other hand, if too few cores are activated, it will create a bottleneck as the Spike-MCryptCores throughput is smaller than the incoming data rate. Therefore, deciding the number of cores to be activated can be crucial.

Fig. 1(b) illustrates the hardware block diagram of the Spike-MCryptCores platform, including (1) *Hardware SNN:* controls the process of splitting and concatenating data (MUX, DEMUX) and on/off AES cores; (2) *DEMUX:* Demultiplexer for input data; (3) *MUX:* Multiplexer for output data; (4) *I-FIFOs and O-FIFOs* are Input Data Buffers and Output Data Buffers, respectively; (5) *AESCs:* consists of *N* AES cores (AESCs) operating in parallel. Note that the number of AES cores (*N*) can be configured at the design phase. With the hardware configurations generated by the Software, the hardware architecture of SNN can perform the same prediction task, which allows it to generate a suitable number of cores to be activated. Since the whole SNN inference can be done on hardware in parallel, the system does not need any dedicated CPU to perform the prediction. Moreover, thanks to the low complexity and low power features of SNN, the whole controller can be power efficient and will not create considerable overhead in terms of power and area cost.

3.2. *Software design for platform*

As described in Section 3.1, Fig. 1(a) is a process diagram of software implementation, consisting of two main stages. The purpose of the software is to generate possible scenarios from the input data. These scenarios are used to train the SNN model and then generate the SNN hardware configuration.

3.2.1. *Stage 1: generate data to train*

As we mentioned above, there are several approaches to training an SNN. In this work, we will train an ANN using supervised learning and convert the ANN to SNN [43]. By using the conversion method, we must prepare the data consisting of inputs and labels for training/testing.

In this section, we discuss in detail how to use the Python programming language to create data scenarios for SNN training. Usually, data can be collected from realistic datasets. However, as it is difficult to manage the quality of data (which could be redundant and randomized), we decided to generate a synthetic dataset for training and testing. Although we use this synthetic data set, designers can collect realistic data rates and use them for training/testing.

Data scenarios transform according to functions such as *exponential, sine, tan, step, sawtooth* functions, and so on (The details of the predefined data scenarios are presented in TABLE A in the Supplemental Document). Please note that designers can add or remove functions to adapt the system to different scenarios. Actual data can also be recorded and used to train the SNN as well. In this work, we also added a random case (see Fig. 4(m)) as an example of user define cases.

First, we prepare the data in the form of given functions. The data rate *(Data(t))* is counted during the sampling time T (T is the number of system clocks per each sampling). T can be changed to suit the design of the system (e.g., $T = 100$, 128, 256, 512, or 1024). Based on the value of *Data(t),* the software will calculate inputs and labels.

Algorithm 1 presents the data generation algorithm for the SNN training process. The inputs of the algorithm are:

- *num_of_core*: number of AES cores;
- *samp_of_data*: number of data samples generated for each scenario;
- *clock_in_sample*: number of system clocks in each sample (T).
- *gen_function*: the function used to generate data.

The outputs of the algorithm are: *Data(t), Data(t-1), Data(t-2), Data (t-3), Delta(t-1), Delta(t-2), Delta(t-3), Residual(t-1), Enable_cores(t)* and *gen_data*. While *gen_data* the binary array indicates the incoming data for Spike-MCryptCores, the other outputs are described in Table 1. Note that the first eight values are the inputs and the last one (*Enable_cores(t)*) is the label for training and testing of the SNN.

To understand how the data is generated, we would like to illustrate the case of the exponential function in Fig. 3. The horizontal axis is time. For convenience of calculation, we choose the sampling period $T = 100$ clock cycles. There are two graphs in the chart. The black graph represents 100 input data samples. The traffic rate is the number of data blocks contained in $T = 100$ clock cycles (one data block is 128 bits). Traffic rate is changed each interval of 100 clock cycles. In this case, the traffic rate is an exponential function. The red graph shows the number of cores that need to be turned on to avoid data loss. In lines 1-3, the Algorithm 1 loops from 0 to samp_of_data-1 and generate a series of *Data (t)* (*y(t)* is a copy of *Data(t)*). The value of *Data(t)* is based on the *gen_function*. For example, if *t=80* the value *Data(t)* is 63 which means in T cycles, the Spike-MCryptCores has 63 incoming data blocks.

From lines 4 to 9 in Algorithm 1, the algorithm generates randomly the data for each T cycles. The *gen_data*$[t][i]$ will be generated randomly between 0 and 1 (0: no incoming data block, 1: has incoming data block). The second loop will break once it reaches *Data*(t) incoming data blocks in T cycles (as $y[t]$ down to zero).

Finally, the algorithm calculates the remained values using the equations in Table 1 (line 9) and returns the calculated values (line 10). The value of enable cores for exponential function can be seen in Fig. 3.

As can be seen in Table 1, Stage 1 basically gives the history of data rates as inputs, and the correct number of cores should be activated as the label. The key idea is to let SNN predict the proper number of cores by knowing the history of the data rate. As the data rates usually go serially by proper patterns (or functions), the SNN can predict the number of cores. If the data rates are randomized, there is less chance that SNN can predict. In this work, we only consider the four previous sampling periods as the history of SNN training. Obviously, we can even extend or reduce the range. Based on our experimental results reported in Section 4.3, having eight inputs for training and testing can obtain more than 95% accuracy with the SNN model of 8-5-11.

Fig. 4 illustrates the data scenarios. Each scenario consists of $sample\_of\_data = 100$, and $clock\_in\_sample = 100$. The *Data (t)* takes values from 0 *to* 100; $enable\_core(t)$ takes values from 0 to $number\_of\_core$ ($number\_of\_core = 10$ is the number of AESCs in the Spike-MCryptCores platform).

In the data set, we have two types of functions. The first type is just the standalone functions as in Fig. 4(a-l). The second type is the combination of the functions and randomization. Fig. 4(m) first shows a random function. Then, we combine the random function with the function in Fig. 4(a-l). As the data rates might vary and we try to mimic those behaviors.

Algorithm 1
Generate data for training

```
1    Input: num_of_core, samp_of_data, clock_in_sample, gen_function
     Output: Data(t), Data(t-1), Data(t-2), Data(t-3), Delta(t-1), Delta(t-2), Delta(t-3),
     Residual(t-1), Enable_cores(t), gen_data
     for t in 0 to samp_of_data-1:
2        y[t] = min(gen_function(i), clock_in_sample);
3        Data(t) =y[t];
4    for t in 0 to samp_of_data-1:
5        for i in 0 to clock_in_sample-1:
6            gen_data[t][i] = randomize(0,1);
7            y[t] = y[t] - gen_data[t][i];
8            if y[t] == 0: break;
9    calculate: Data(t-1), Data(t-2), Data(t-3), Delta(t-1), Delta(t-2), Delta(t-3),
     Residual(t-1), Enable_cores(t) from equation in TABLE 1.
10   return
```

3.2.2. Stage 2: SNN network models and learning algorithm

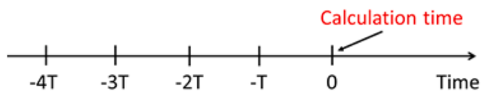
With the data rate scenarios generated as in Stage 1, we can use the inputs and labels to train our SNN network. A normal SNN network is defined by the neuron models, the learning rules for the synapse, and the overall topology for the connection between layers of neurons [47]. In this work, we use a feedforward, fully-connected topology of neurons with one hidden layer and one readout layer to determine the required number of AES cores for operations, as shown in Fig. 5. The hidden layer has M neurons, while the readout layer has 11 neurons to represent the 11 possible cases for the number of activating cores (0 to 10). The number of hidden layers can be increased to have a more complex model.

However, as we aim to maintain a low-cost SNN controller and a single hidden layer can provide good accuracy, the SNN model will only use a single hidden layer. Spike-MCryptCores also supports creating different numbers of neurons in a hidden layer and changing the number of hidden layers.

Fig. 6 depicts a basic Leaky Integrate-and-Fire (LIF) neuron, consisting of its synapses, soma, and axon. The synapses serve as the link between two neurons. When a neuron receives input spikes from neurons in the preceding layer, the input spikes are scaled according to the weighted synapse strength, and the weighted inputs are incorporated into the membrane potential at the soma of the neuron. Once the membrane potential exceeds a predefined threshold, the neuron will fire and create an output spike, which is then communicated to the neuron in the next layer through the axon.

We select the conductance-based Leaky-Integrate-and-Fire (LIF) neuron's models since it has low complexity while still maintaining effective computation capabilities. For the hidden layer, the neurons have analog current input. The neurons in both layers used a reset-by-subtraction method. The dynamics of the LIF neurons are captured in Eq. (1).

Table 1
Calculation training data.

Input	Calculation
	
$Data(t-1)$	Data count in $[-T+1{,}0]$
$Data(t-2)$	Data count in $[-2T+1,-T]$
$Data(t-3)$	Data count in $[-3T+1,-2T]$
$Data(t-4)$	Data count in $[-4T+1,-3T]$
$Delta(t-1)$	$= Data(t-1) - Data(t-2)$
$Delta(t-2)$	$= Data(t-2) - Data(t-3)$
$Delta(t-3)$	$= Data(t-3) - Data(t-4)$
$Residual(t-1)$	$= Data(t-1) - Throughput(t-1)$
$Enable\_cores(t)$	Number of enabled cores for $[0,T]$ (Labels) $Enable\_cores(t) = (Data(t) + 0.5 * T/N)/N$

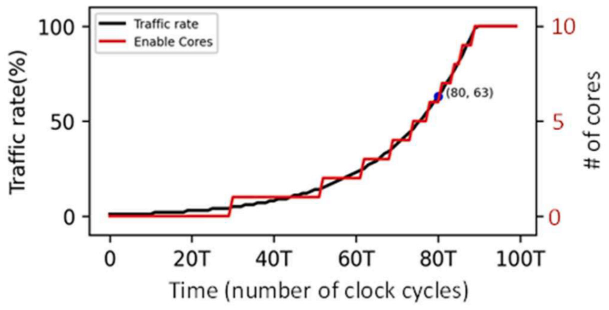


Fig. 3. An example of data scenarios: exponential function.

$$V_i^l[t]_{pre} = \sum_j w_{i,j}^l \times I_j[t] \times O_j^{l-1}[t] + V_i^l[t-1]_{post} + b_i^l$$

$$O_j^l[t] = \begin{cases} 1 & f\ V_i^l[t] \geq V_{threshold} \\ 0 & otherwise \end{cases} \tag{1}$$

$$V_i^l[t]_{post} = \begin{cases} V_i^l[t]_{pre} & if\ O_j^l[t] = 1 \\ V_i^l[t]_{pre} - V_{threshold} & if\ O_j^l[t] = 0 \end{cases}$$

Where $V_i^l[t]_{pre}$and $V_i^l[t]_{post}$denote the pre and post-fire membrane potential of neuron-i at layer l at timestep t, while $w_{i,j}^l$ denotes the synaptic weight between the pre-synaptic neuron-j and the post-synaptic neuron-i, and b_i^l is the bias term for neuron-i at each layer l. Each neuron will integrate the product of $w_{i,j}^l$ and the input current $I_j[t]$ from all neurons in the previous layer. The bias term is added after the integration process is completed. A neuron will emit a spike $O_j^l[t]$ if the pre-fire membrane potential crosses a certain threshold $V_{threshold}$. After firing, the neuron will reset to the post-fire potential by subtracting $V_{threshold}$. The output spike is sent to the downstream layer.

This work adopts the ANN-to-SNN conversion technique depicted in [42] to train the network. The trained network then will go through a Post-Training-Quantization (PTQ) process before being deployed with the hardware model. The main reason why we would like to perform quantization is that reducing to a fixed-point format could lead to less hardware complexity.

An overview of the conversion technique and the PTQ process is shown in Algorithm 2. The conversion from ANN-to-SNN starts with the training of a fully-connected ANN with backpropagation. A weight normalization process is then carried out with part of the training data to ensure the ratio between the weights and the threshold is kept in balance. For a detailed discussion on the weight normalization process, we refer to the original paper in [48].

The trained SNN networks need to be quantized to fixed point format *Qi.f* before the hardware evaluation step. The *Qi.f* fixed point format used 1 sign bit, *i* bits for the integer part, and *f* bits for the fractional part. It is able to represent numbers in the range of $[-2^i; 2^i - 2^f]$ with a precision of $\epsilon = 2^{-f}$. We round the floating-point number x to its fixed-point counterpart by the rounding to the nearest method, as depicted in Equation (2):

$$Round(x) = \begin{cases} \lfloor x \rfloor & if\ \lfloor x \rfloor \leq x < \lfloor x \rfloor + \frac{\epsilon}{2} \\ \lfloor x \rfloor + \epsilon & if\ \lfloor x \rfloor + \frac{\epsilon}{2} \leq x < \lfloor x \rfloor + \epsilon \end{cases} \tag{2}$$

On this platform, we not only design the SNN model in software, but also design the hardware architecture of this SNN to be integrated into the controller of the Spike-MCryptCores. Details on the hardware architecture of SNN are shown in Section 3.3.

3.3. Hardware architecture of the platform

In Section 3.2, we have illustrated how we create synthetic data and train the SNN model. The hardware architecture for the SNN is also proposed. In this section, we will describe the hardware architecture of Spike-MCryptCores.

Fig. 7 presents the detailed hardware architecture of the Spike-MCryptCores platform, which includes the following modules:

- DEMUX: A demultiplexer with 1 input, *N* outputs, and DSEL control signal. DEMUX is used to distribute the input data among each module.
- I-FIFO: Input data buffers for AESCs. There are *N* I-FIFOs in the hardware of the Spike-MCryptCores platform.
- AESC: There are *N* AESCs in the hardware of the Spike-MCryptCores platform.
- Expand Key: The module generates subkeys for AESCs.
- O-FIFO: Output data buffers for AESCs. There are *N* O-FIFOs in the hardware of the Spike-MCryptCores platform.
- MUX: A multiplexer with *N* inputs, 1 output, and MSEL control signals. The MUX module is used to merge all the data from all the AES cores into a single stream. The MSEL selection signal relies on DSEL which allows it to keep the order of data.

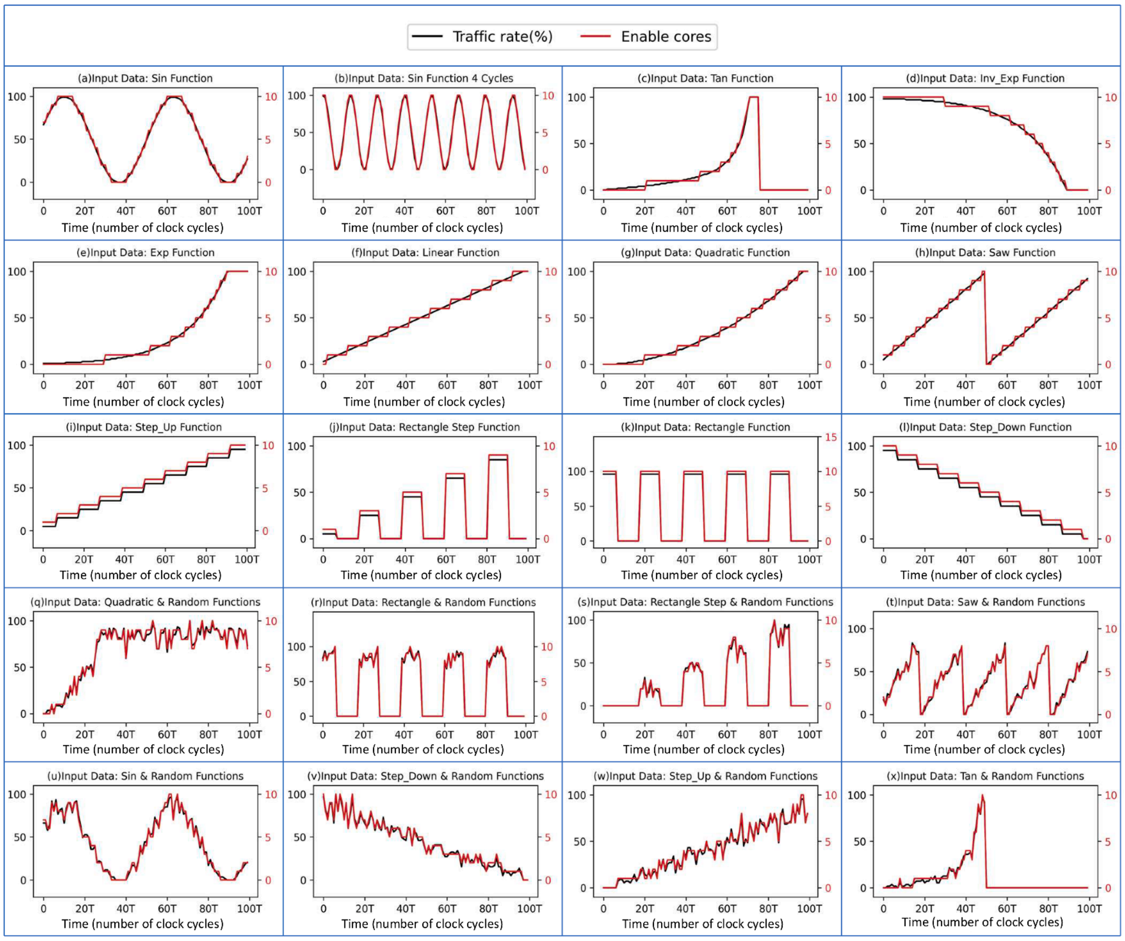


Fig. 4. Data scenarios used during SNN training.

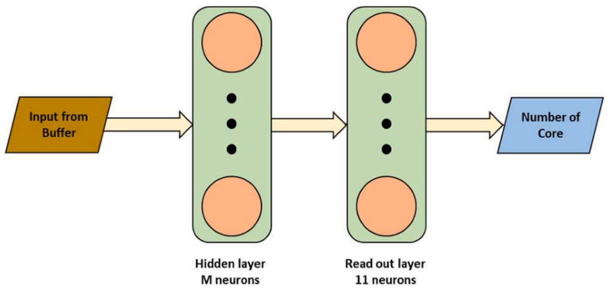


Fig. 5. Our SNN network with the feedforward topology.

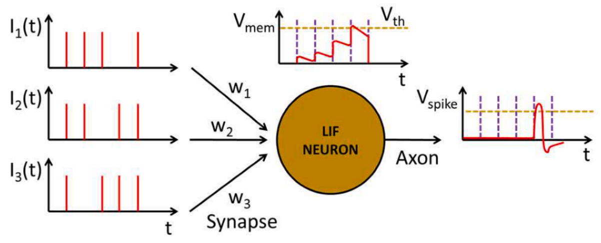


Fig. 6. Leaky integrate-and-fire (LIF) neuron model.

Algorithm 2
SNN network models and learning algorithm

```
        Input: Data training (.csv file)
        Input: Labels (.csv file)
        Output: Hyperparameters, weights, bias
        Begin
1               S1: Train the ANN model.
2               S2: Weight Normalization
3               S3: Convert to SNN Network
4               S4: Quantize the SNN Parameters
5               S5: Quantized SNN Network (Qi.f)
6               S6: Testing for SNN Accuracy
7               S7: Calculate Hyperparameters, weights, and bias
        End
```

- SNN Controller: This module will perform the prediction of the number of cores to be turned on/off (by gating the clock of each AES core). The module also generates control signals for DEMUX, I-FIFO, AESC, O-FIFO, and MUX.

We would like to note that the Spike-MCryptCores platform operates in two frequency domains. Sys_Clk is the clock signal of the system, while Core_Clk is the clock signal of the AESCs. By having two clock domains, the AES core can operate at a lower frequency while having N cores can still provide high throughput to the system. Here, we fixed $Sys\_Clk = N \times Core\_Clk$ to balance the throughput at the maximum rate cases. I-FIFOs and O-FIFOs operate on both Sys_Clk and Core_Clk frequency domains.

Data_in has a data width of 128 bits. An example of input data has a waveform as shown in Fig. 8. Here, there are times when there are a lot

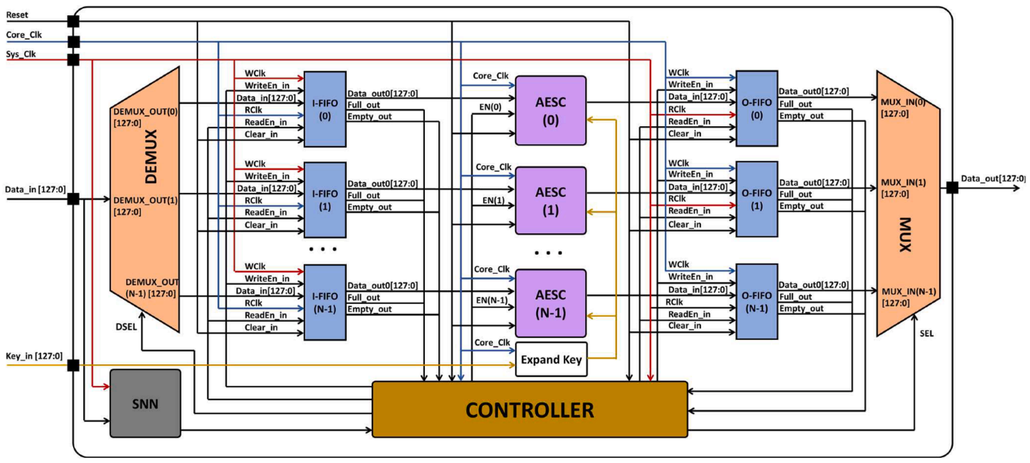


Fig. 7. Hardware architecture of Spike-MCryptCores platform.

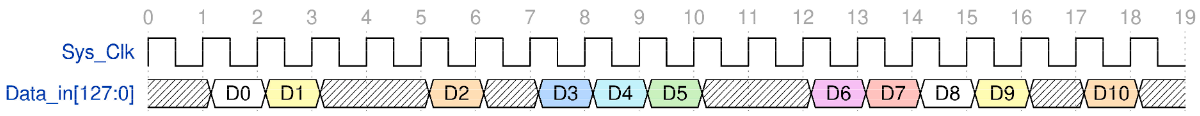


Fig. 8. Example of waveform graph of input data.

of inputs, there are times when there is no input. For example, at the clocks 1, 2, 5, 7 to 9, 12 to 16, 17, there is input data. At the clocks 0, 3, 4, 6, 10, 11, 16, 19, there is no data to the system. Therefore, the input data rate can be different in scenarios for the same function.

I-FIFOs and O-FIFOs, which operate on two frequency domains (Asynchronous FIFO), are data buffers that handle the conversion between two clock domains (*Core_Clk* and *Sys_Clk*). Because AESCs operate in parallel, their operating frequency is slower than that of the system (in this work, we defined $Core\_Clk = Sys\_Clk/N$). I-FIFOs convert input data from DEMUX (high speed) to AESCs (low speed). In contrast, O-FIFOs convert data from AESCs (low speed) to MUX sets (high speed). By having a depth of four data blocks, I-FIFOs and O-FIFOs help avoid data loss due to delays during control with the SNN Controller.

3.3.1. SNN controler

In the previous sections, we presented the architecture for Spike-MCryptCores with N AES cores. As the Spike-MCryptCores hardware allows to turn on and turn off the clock signals to each of its AES cores, the system needs to control the enable signal of each core. Here, we designed the SNN controller with the ability to predict the suitable number of cores and provide enable signals to each AES core. SNN's hyperparameters, weights, and biases are trained in the software model.

Algorithm 3 presents our proposed algorithm for the SNN Controller. The algorithm consists of six phases: **S1:** counting data; **S2:** storing values; **S3:** calculating parameters; **S4:** calculating the number of AESCs to be turned on; **S5:** controlling MUXs, DEMUX FIFOs; and **S6:** controlling the AESCs. The block diagram for the controller is shown in Fig. 9.

- **S1** - Counting input data: Input data valid signal *(Data_in_valid)* is put into Counter to count the number of data occurrences in time of *T* cycles (*T* = *100, 128, 256, 512, 1024 Sys_Clock)*. The number of data occurrences in period *T* is *Data(t)*.
- **S2** - Storing values: *Data(t-1), Data(t-2), Data(t-3), Data(t-4)* values are saved to calculate the parameters in S3.
- **S3** - Calculating the change of data at time *t-1, t-2,* and *t-3* by the corresponding equations in Table 1.
- **S4** - Providing Inputs for hardware SNN. Based on the input values, SNN calculates #Core - which is the number of AESCs that need to be turned on at time *T*. At the end of phase **S4,** the SNN outputs the number of cores being turned on.
- **S5+S6** - SNN Controller uses the #Core value and the 'Empty', 'Full' signals of the I-FIFOs and O-FIFOs to output the DSEL signals that

Algorithm 3
SNN controller

```
        Input: data_in, Empty, Full
        Output: #core
        Output: DSEL, MSEL
        Output: EN(0), EN(1),..., EN(N-1)
        Begin
1           S1: Count for Data(t) each T cycles.
2           S2: Store Data(t), Data(t-1), Data(t-2), Data(t-3), Data(t-4).
3           S3: Calculate Delta(t-1), Delta(t-2),
            Delta(t-3), Residual(t-1).
4           S4: Predict the number of cores being turned on by SNN hardware.
5           S5: Generate DSEL, MSEL signals
6           S6: Generate EN(0), EN(1),..., EN(N-1).
        End
```

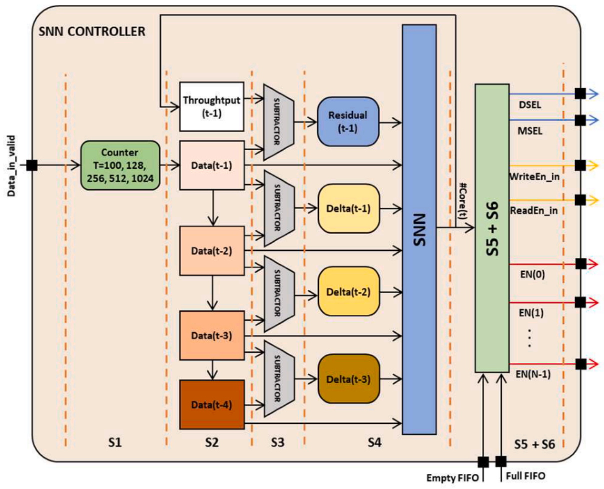


Fig. 9. SNN controller diagram.

control the DEMUXs and the MSELs that control the MUXs. The controller also computes and outputs the WriteEn_in and ReadEn_in signals to control the reading and writing of I-FIFOs and O-FIFOs. On the other hand, the *En(0)* to *En(N-1)* signals are also generated by the SNN Controller based on the number of cores to control the on/off clock of the AESC(0) to AESC(N-1) cores.

Fig. 9 illustrates the architecture of the SNN controller. The data rate is counted by a counter each *T* cycles. Then, the data rate is stored in the register of *Data(t-1), Data(t-2), Data(t-3), Data(t-4)*. The delta value is calculated by subtracting a pair of data rate values. The eight inputs are fed into SNN to predict the number of cores. As the input vector must be normalized, here we counted for $T =$ *power of two cycles* (i.e., 128, 256, 512) which allows the normalization converts to shift bit function. In the other words, we can eliminate the normalization function.

The detailed architecture of the SNN is shown in Section 3.3.2. After having the number of cores being turned on, the sub-controller will generate the Enable signal and the DSEL/MSEL signal as in **S5** and **S6**.

3.3.2. *Hardware architecture for the SNN's block*

In Section 3.2, we have illustrated the data generations and how we choose and train the SNN model. Obviously, the SNN can be performed by a dedicated CPU within the system; however, it will introduce a significant amount of area overhead. In our proposed Spike-MCryptCores, we use a hardware SNN architecture to compute. We also already quantized the SNN model to be ready for the hardware SNN.

Fig. 10 shows the hardware architecture for a single Processing Element (PE) and the block diagram for the SNN network. The dynamics of the LIF neurons are handled by the PE. Each PE consists of a simple Multiply-and-Accumulate (MAC) which will integrate the inputs to the neuron in each time step. The weights and the input current are kept at 8-b precision. It depends on the mode of operations, the PE could integrate the products of the input current and the weights (in the case of analog input), or with only the weights (in the case of binary spiking input). A comparator is used to give an output spike.

In conventional SNN architectures [37–39], the Leaky-Integrate-and-Fire neuron is usually processed with binary inputs (spikes). However, in this work, we decided to use the input current in 8-bit format and MAC to compute. The main reason is to reduce the complexity of the design. If the design follows the spike-based approach, the inputs (data and delta values) must be converted to spikes. While in software this could be done easily, in hardware it requires a pseudo-random module (i.e., a linear feedback-shift register and a comparator) for each input, which significantly requires more area cost. In this work, we used currents as the input to reduce the pseudorandom module for the first layer. For the following layers, we use spikes for computation.

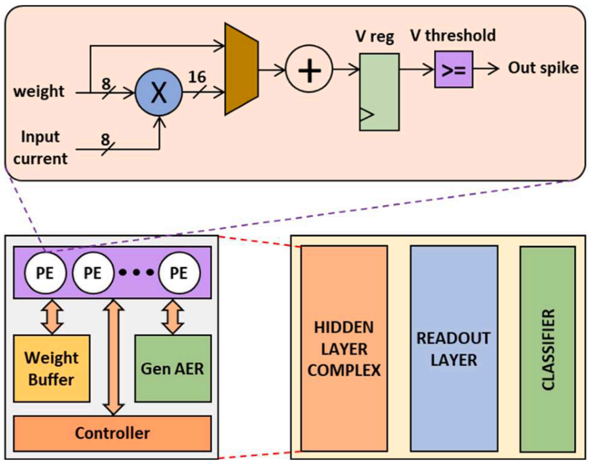


Fig. 10. Hardware architecture for the SNN system.

This type of hybrid design can have a lower area cost and lower latency.

Neurons in the same layer are handled by the same PEs complex. The number of PEs complexes and the number of neurons in each complex are fixed by the network architecture. Each complex has a dedicated SRAM buffer to store the trained SNN's parameters such as weights and bias. After each timestep, the output spikes are encoded into the Address-Event-Representation (AER) format, which is a popular encoding format for spikes in modern neuromorphic formats. With AER, the address of each neuron is encoded to send to the next layer. In the next layer, the incoming AER signals are used to load the correct weights for integration. The layers communicate with each other through a simple handshake protocol to ensure operations can be handled in a pipelined, sequential fashion.

The details of DEMUX, MUX, I-FIFO, O-FIFO, AESC, and ExpandKey are shown in the Supplemental document.

4. Results

In this section, we present the evaluation results of the Spike-MCryptCores platform. We first describe the evaluation methodology. Then, we present the hardware results in CMOS 45nm technology. In the following part, we evaluate the training results with the SNN model. The power consumption of the Spike-MCryptCores hardware is also compared with that of the MCryptCores to highlight the benefits of the SNN Controller. The hardware architecture of MCryptCores basically consists of *N* AES cores without a clock gating feature and an SNN controller.

4.1. *Evaluation methodology*

To generate data for SNN training, we use Algorithm 1 to generate different data scenarios using the Python programming language. Input data $Data(t)$ are generated according to the scenarios of functions. The algorithm then computes the Inputs and Labels as shown in Table 1. Finally, the Inputs and Labels are saved to files for the training process.

For training and converting, we first train a fully-connected ANN with backpropagation using the SpikingJelly framework [49]. A weight normalization process is then carried out with part of the training data to ensure the ratio between the weights and the threshold is kept in balance. The trained network then goes through a Post-Training-Quantization (PTQ) process before being deployed with the hardware model. Weights and biases are downloaded to SNN hardware later at the beginning of its operation.

For the hardware, we evaluate the hardware implementation results of the Spike-MCryptCores architecture, such as area cost, power consumption, and layout. To highlight the effectiveness of the SNN Controller, we evaluated 24 data samples as shown in Fig. 4.

In this work, we decide to have $N = 10$ core AES within the Spike-MCryptCores. We pick it as a case study and obviously the system can adapt to different numbers of cores.

4.2. *Hardware evaluation results*

The hardware architecture of Spike-MCryptCores is designed in VHDL, simulated, and verified on ModelSim. We synthesize and analyze power consumption and layout with Synopsys Design Compile, Prime

Table 2
Hardware complexity of the spike-MCryptCores.

Module	Absolute Total (mm^2)	Percent (%)
Spike-MCryptCores	0.992	100
10 AES cores	0.898	90.5
DEMUX + I-FIFO	0.036	3.6
MUX + O-FIFO	0.036	3.6
SNN CONTROLLER	0.021	2.3

Time, and Cadence Innovus using the CMOS NANGATE 45 nm library.

Table 2 presents the hardware complexity of the modules in the Spike-MCryptCores Platform. With an $N = 10$ AESCs configuration, the area cost of the Spike-MCryptCores platform is 0.992 mm^2. In which the area cost for AESCs is 0.898 mm^2, accounting for 90.5 %. Hardware costs for DEMUX and MUX modules account for 7.2 %. While the SNN controller has only 2.3 % of the total. It can be seen that with a very small area cost, the SNN controller can predict the number of cores and can turn on/off the clocks of the AESCs in accordance with the incoming data rate.

Table 3 shows a summary of the power consumption of the Spike-MCryptCores architecture using clock gating and the MCryptCores architecture not using clock gating. Both architectures operate at a clock frequency of 50 MHz. In the absence of encrypted input, the SNN Controller in the Spike-McryptCores architecture turns off the clock to all AESCs. For the MCryptCores architecture, the clocks of the AESCs are not disconnected. In this case, the power consumption of Spike-MCryptCores is 24.5 mW and that of MCryptCores is 104.7 mW. Thus, the power consumption of MCryptCores is 4 times higher than that of Spike-MCryptCores. In case of the maximum incoming bandwidth, all cores are enabled, the power consumption of Spike-MCryptCores is 173.7 mW and that of MCryptCores is 253.1 mW. In this case, MCryptCore's power consumption is still 1.4 times higher than Spike-MCryptCores. This power reduction is thanks to the intensive clock gating in the Spike-MCryptCores. In summary, the Spike-MCryptCores architecture using clock gating technology can save from 31.4 to 76.6 % power consumption compared to the MCryptCores architecture without clock gating technique.

Fig. 11 is the Layout and Floorplan of Spike-MCryptCores with dimensions of 1200 × . 1425 µm^2 consisting of the following main modules: AESCs (10 AESCs) that occupy the majority of the chip area (90.5 %), DEMUX and MUX that take up 7.2 %, and the rest is the SSN Controller, accounting for only a small part (2.3 %) of the chip.

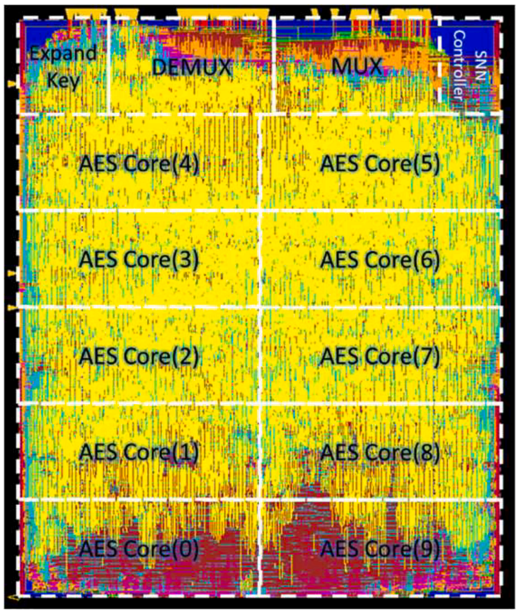


Fig. 11. Spike-MCryptCores layout & floorplan

Table 4
Training results.

Network	8-3-11	8-5-11	8-10-11	8-15-11
ANN	95.82%	98.326%	95.81%	100%
SNN controller (32-bit)	89.29%	97.72%	95.67%	97.72%
SNN controller (8-bit)	89.29%	95.44%	96.58%	97.27%
Min Diff. (Prediction vs Label)	-1	-1	-1	-1
Max Diff. (Prediction vs Label)	+1	+1	+1	+1

4.3. Evaluation of training results

The data prepared for the training process consists of 2500 data samples and labels. This data is divided into 2 parts. Part 1: randomly selecting 400 samples to test the accuracy of the training model. Part 2: including the remaining 2100 samples used for training.

First, the data samples are trained with a fully connected (floating point) ANN model. In this process, we went through an empirical process to choose the best ANN model to use. The training results with the ANN model are converted to SNN (floating-point). To reduce the complexity when transferring the model to hardware, perform Quantization for quantization (8 bits). Finally, the SNN model (8 bits) is converted to hardware SNN.

Training results with different configurations such as 8-3-11, 8-5-11, 8-10-11, and 8-15-11 are presented in Table 4. It can be seen that the accuracy of the SNN (quantized) is directly proportional to the number of neurons in the hidden layer. With the SNN configuration with 3 hidden layers (8-3-11), the accuracy of the SNN is the lowest (89.29 %). With the SNN configuration with 15 neurons at the hidden layer (8-15-11), the accuracy is up to 97.27 %. However, when increasing the number of neurons in the hidden layer, the hardware complexity of the SNN also increases. Therefore, in this work, we chose the neural network configuration as 8-5-11 to balance the accuracy and the complexity in hardware implementation.

As the training accuracy is not 100 %, we consider the difference

Table 3
Implementation results of spike-MCryptCores and MCryptCores on 45nm CMOS technology.

Active cores		Spike-MCryptCores with Clock gating			MCryptCores without Clock gating			Saving Power (%)
	CLK (MHz)	Total Power (mW)	Throughput (Gbps)	Energy Efficiency (Gbps/W)	Total Power (mW)	Throughput (Gbps)	Energy Efficiency (Gbps/W)	
No core	50	24.5	0	0	104.7	0	0	76.6
One core	50	39.9	6.4	160.3	119.5	6.4	53.6	66.6
Two cores	50	54.1	12.8	236.6	134.3	12.8	95.3	59.7
Three cores	50	69.0	19.2	278.2	149.2	19.2	128.7	53.7
Four cores	50	83.9	25.6	305.1	164.1	25.6	156.0	48.9
Five cores	50	98.7	32.0	324.1	178.9	32.0	165.2	44.8
Six cores	50	113.6	38.4	338.2	193.7	38.4	184.1	41.4
Seven cores	50	128.5	44.8	348.6	208.6	44.8	200.2	38.4
Eight cores	50	143.5	51.2	356.8	223.7	51.2	228.8	35.9
Nine cores	50	158.3	57.6	363.9	238.6	57.6	241.4	33.7
Ten cores	50	173.7	64.0	368.4	253.1	64.0	252.8	31.4

between label and prediction of the SNN Controller (min & max diff) as the key factor. According Table 4, the accuracy of the SNN controller (8-bit) with configuration 8-5-11 is 95.44 %, thus the error rate of this model is 4.56 %. However, in case the prediction is wrong, the difference between the label and the prediction is only 1 unit. That is, if the model predicts incorrectly, the deviation is also very small (the SNN controller predicts less than one core or more than one core in comparison to the label). The Spike-AES hardware has I-FIFO and O-FIFO modules at the inputs and outputs of the AESCs, so data can be stabilized, and we can expect small bottlenecks. Models 8-10-11 or models 8-15-11 can be selected to improve accuracy to 96.58 % or 97.27 %; however, as the hardware complexity also increases, we decide to use 8-5-11 as it gives the best trade-off between accuracy and area cost.

Fig. 12 is a graph showing the results of training with the 8-5-11 neural network model. The model has 8 inputs, $M = 5$ hidden layers, and 11 outputs. With the ANN model, the accuracy reaches 98.326 %, with the SNN model the accuracy increases with timestep T and reaches saturation with an accuracy of up to 100 % at $T = 16$. Please note that since the spikes generated during the training of the SNN are randomly generated using Poisson process, the accuracy may vary during the inference time and the variation can be different by using different seeds for random. We note that the peak at 100 % can be a random noise as it drops lower after 17 timesteps. In general, we only consider the final results as the controller only considers them.

Fig. 13 illustrates the test result with random values (1-100 in 2500). We generated 2500 random data samples and tested them with the trained model 8-5-11. The prediction accuracy of the SNN controller is only 560/2500 (22.4 %). This result is predictable as the SNN cannot deal with this type of change. It can be seen that the prediction graph of the SNN controller is different from the ideal graph. Out of 2500 data samples, 560 (22.4 %) samples are predicted as the same as the ideal value, 1179 samples end up with one core difference, and 761 samples have 2+ cores difference.

4.4. Evaluation of SNN controller performance

Since the Spike-MCryptCores utilize multiple AES cores with the ability to turn off the clock signal of the unused core using an SNN controller, it can significantly reduce the power consumption. On the other hand, MCryptCores does not have a control mechanism to turn off the clock signal of the unused core, it still consumes dynamic power from the clock signals. To compare the power consumption of Spike-MCryptCores vs MCryptCores (multi-AES cores with clock-gating and SNN controller), we tested both platforms with the same input dataset of 24 scenarios. The results are shown in Fig. 15. Note that, besides the overall power consumption, we also evaluate the residual value of the Spike-MCryptCores. The residual will reflect how well the SNN controller adapts to the scenarios. If the residual is negative, it means there is an unused core. If the residual is positive, it means there are

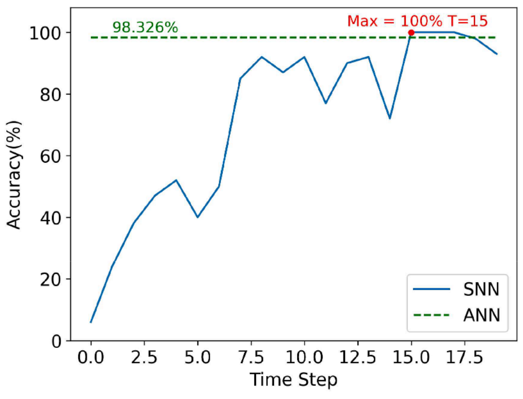


Fig. 12. ANN and SNN Accuracy using 8-5-11 configuration.

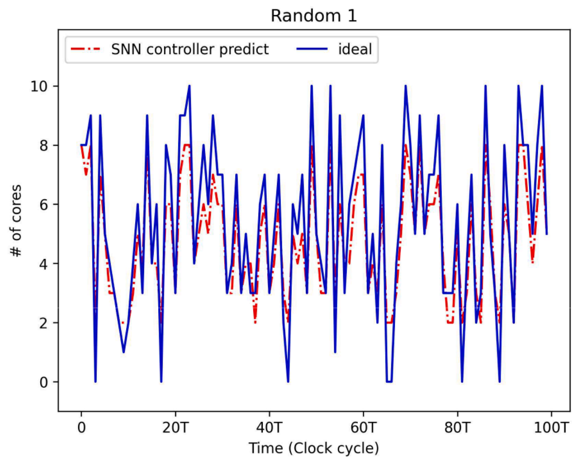


Fig. 13. SNN control test result with random values (1 to 100 in 2500).

residual data blocks not being processed after T cycles.

In the scenario where the input data is a sine function (Fig. 15(a)), the power dissipation of the Spike-MCryptCores and MCryptCores are also sine functions and correspond to the input data. However, the average power consumption of Spike-MCryptCores (99.24 mW) is lower than the average power consumption of MCryptCores (176.26 mW). Thus, with the same sine data scenario, Spike-MCryptCores saves 45 % power compared to MCryptCores. Same with other scenarios in Fig. 15. Power consumption of Spike-MCryptCores is 39 % to 67 % lower than MCryptCores.

In Fig. 15 (a) and (l), Spike-MCryptCores residual is -5 in most samples, indicating that switching AESCs is not optimal, AESCs have not used up their throughput. In Fig. 15(j) and (k), the Spike-MCryptCores residual is zero in most samples, indicating that switching AESCs is optimal, AESCs using their throughput to the maximum. In all scenarios in Fig. 15, the data residual varies from -5 to +5. Although there is an unused core or residual data block, the value is relatively small (maximum 5 unused cycle cores or 5 residual data blocks in total $N*T = 1000$ cycle cores).

In Fig. 14, we present the average power of Spike-MCryptCores and MCryptCores for all scenarios. Accordingly, the most evaluated data scenario (best case) is the tan function. In this scenario, the power consumption of the Spike-MCryptCores is 39.78 mW, while the power consumption of the MCryptCores is 119.84 mW. Thus, with the same data scenario as the tan function, the power consumption of Spike-MCryptCores is only 33 % of that of MCryptCores. The worst-case data scenario is the *square_rand* function. In this scenario, the power consumption of the Spike-MCryptCores is 126.86 mW, while the power consumption of the MCryptCores is 206.95 mW. Thus, in this scenario, the power consumption of Spike-MCryptCores is equal to 61 % of MCryptCores.

The average power consumption of Spike-MCryptCores in 24 data scenarios is 84.85 mW, while with MCryptCores is 164.93 mW. Thus, the average Power Consumption of Spike-MCryptCores in 24 data scenarios is equal to 51.4 % of MCryptCores. Spike-MCryptCores achieves controllability as expected. The control accuracy is up to 95.44 %. In 4.56 % of the error samples, the difference is ±1 number of cores. In the above scenarios, the scenario with a low data rate (e.g., a tan function scenario) is highly energy efficient. In the best case (tan scenario), the power consumption of Spike-MCryptCores is only 33 % of that of MCryptCores. The average of 24 Spike-MCryptCores scenarios saves up to 51.4 % energy compared to MCryptCores.

Table 5 shows the comparison between our method and other low-power solution for System-on-Chip. Here, our method achieves a maximum of 67.0 % while other DVFS can only achieve up to 51 %. Work in [49] also uses clock gating and obtains a comparable result. In

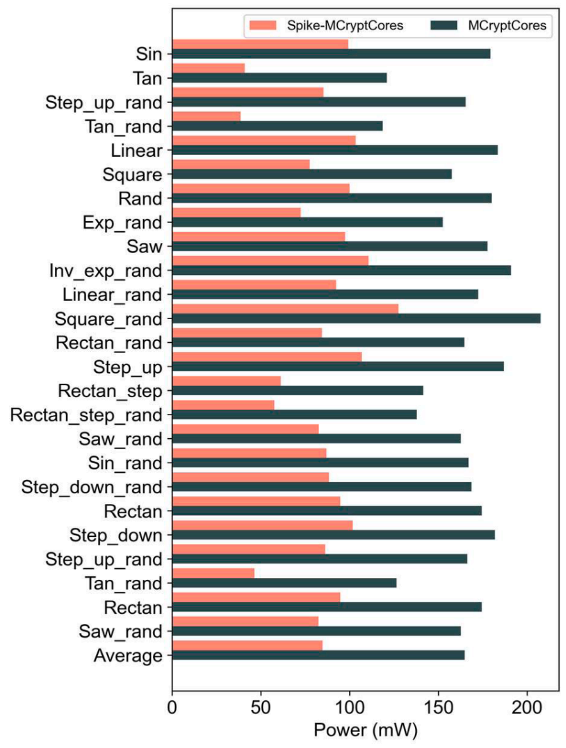


Fig. 14. Compare the average power consumption of Spike-MCryptCores and MCryptCores.

summary, our platform shows an approach to reducing power consumption with comparable efficiency. Please also note that in comparison to other methods, SNN allows more flexibility with the ability to retrain to adapt to new scenarios.

5. Discussion

In the previous sections, we have shown the Spike-MCryptCores platform with the ability to train and adapt to different data rates to optimize the power consumption of the system. Although the results show Spike-MCryptCores can decrease from 39% to 67% of the power consumption, there are some limitations of the system we would like to point out.

First, the system relies on offline training and still cannot perform adaption during its operation (i.e., realize the underperformed cases and adapt the SNN). This is due to the lack of an online training algorithm for multiple layer perceptron for SNN. If designers would like to have online training, there are two solutions: (1) use a dedicated CPU to collect data and train the SNN, and (2) use a bio-plausible Spike-Timing Dependent Plasticity (STDP) online learning. Although training is computation and memory intensive, with a small model like ours, it is totally possible to train. Our open-source training program on Google Collab takes around 20 s to train our model with 100 epochs. Therefore, it is possible to train the system once it finds out that the SNN cannot deal with the data rates. The second approach is of course to adopt a bio-plausible learning algorithm such as STDP [37]. However, STDP learning is a local learning approach and usually works with a single layer. In order to adapt to different scenarios, a single layer of a plural number of neurons with lateral inhibitory can be used. Once a pattern is recognized by one of the neurons due to its compatible weight, it issues inhibitory spikes to other neurons which further allows it to continue to fire [41]. By indicating the most firing neuron, the system can indicate the possible control method. This could be one of the future works for our group to provide the ability to adjust the system online.

Second, the Spike-MCryptCores platform is still not optimal in terms of residual status. This could be easily explained by the fact that the control is not perfect, and the system needs to rely on input and output buffers to stabilize the connection as the data input varies randomly. However, as we tested with 24 cases, there was no data loss, and the Spike-MCryptCores still reduced the power consumption significantly. One of the potential solutions for this is to provide handshaking between the Spike-MCryptCores and the module that sends the data. This could help eliminate the Input and Output FIFO. However, it may lead to long latencies and create a chain of postponement in computation/communication. Data dependencies could be problematic as the output of Spike-MCryptCores is needed for the other tasks.

Third, we can easily realize that despite the Spike-MCryptCores platform being designed for multiple AES cores, the method can be applied for other computing applications as long as the cores are identical and exchangeable. However, there are two main reasons why AES computation can be treated differently. The first reason is that the encoding process for AES is separated for each 128-bit. For other applications, the tasks could be dependent on each other and could lead to idle cores under load. Adapting this to other applications must be investigated carefully. The second reason is the high complexity of AES. As shown in the hardware complexity results, the 10 core AES takes nearly 1 mm^2, which allows us to have flexibility in designing SNN (2.3 % of the whole area cost). Because of these two reasons, we believe the AES applications can be engineered to fit with the bio-inspired controller. This work could be a pilot for other works on how we use SNN for controlling on-chip applications.

Forth, although the clock-gating may reduce up to 67 % of the power consumption, we are aware of the other low power techniques such as power gating or dynamic voltage-frequency scaling (DVFS). The power gating can be used as it cuts off the power of the module, and we can use the SNN control for this approach. DVFS can provide more choices for us to adapt to the system. Both of the approaches are considered future works, and we believe the SNN can adapt to them.

Fifth, the prepared data set in this work is synthetic and we are aware of the possibility of having realistic cases that are not in our scenarios. However, the SNN can be easily extended with new data in the data set, and we already provide random cases for the training. Since the software platform is open source, designers can adapt the method to other data rates and scenarios.

Sixth, while implementing the multi-core system like Spike-MCryptCores, we use MUX and DEMUX to distribute the data and collect the encoded data from each core. However, this approach can be limited due to the fan-out of the DEMUX. Alternatively, there are several communication paradigms to help communicate between cores, like buses or Network-on-Chips (NoCs). Work in [14] utilizes an asynchronous Network-on-Chip to deliver its data flow. Apparently, adopting NoCs can enhance the scalability of our system. Our research group has been working on NoCs [55] and this is one of our future research projects.

Seventh, in this work, we use a feedforward, fully-connected topology without exploiting the sparsity in the connections. For more complicated models, exploiting the sparsity to reduce the complexity of the SNN can be important; however, since our model is lightweight and only take 2.3 % of the area cost, exploiting the sparsity connection will be future work.

Eighth, synchronous communication is used between the layers of the SNN of this work. However, using asynchronicity in communication can also be utilized [35,36]. Regardless of the design choice, this work proves the idea of using SNN can be applied, and using synchronous or asynchronous communication will not affect the overall accuracy.

Although Spike-MCryptCores has the above limitations, the evaluation still shows that it is extremely efficient in power consumption.

6. Conclusion

In this paper, we have proposed Spike-MCryptCores platform – a low

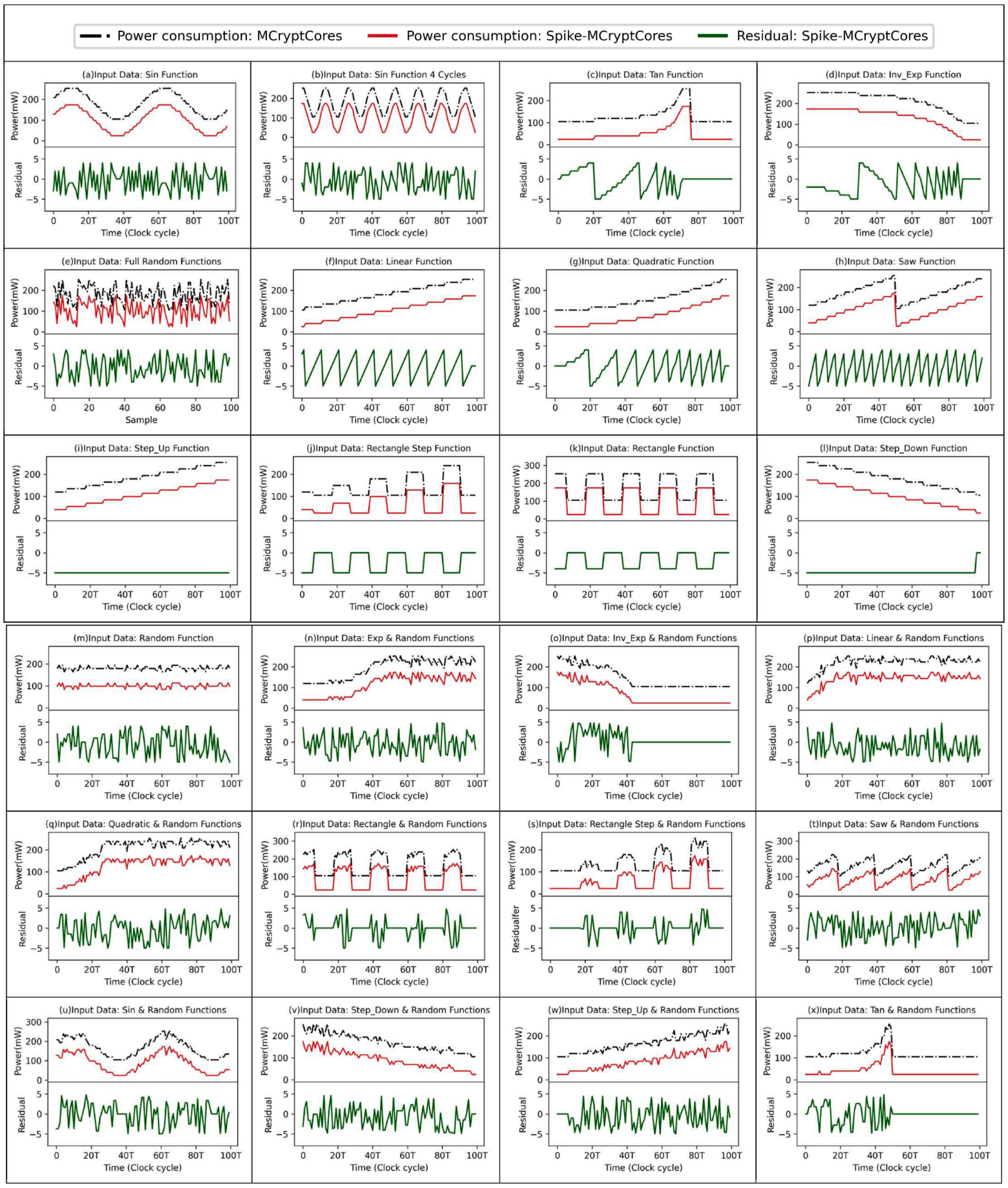


Fig. 15. Spike-MCryptCores vs MCryptCores power consumption comparison with different data scenarios.

power multi-core AES platform with a neuromorphic controller. Spike-MCryptCores consists of a software part that allows designing, training, and testing SNNs for controller and a hardware part that consists of multiple AES cores controlled by the SNN hardware counterpart. The software architecture for SNN has successfully trained with more than 95 % accuracy with only a single hidden layer of 5 neurons and only one core difference from the label in the error cases. The proposed platform only has 7.6 % area overhead for multiplexer, demultiplexer, and asynchronous FIFO. Furthermore, the SNN controller only occupies 2.3 % of the area of the system, which is insignificant. With the SNN controller, the system can reduce power consumption by 67 to 39 % in comparison to the parallel AES core. In summary, the Spike-MCryptCores has introduced a novel method to design and control multiple-core systems with extremely small overhead and high accuracy.

In future works, we would like to adapt the Spike-MCryptCores to different types of multi-core applications. Furthermore, extending Spike-MCryptCores with other low power techniques such as power gating or dynamic voltage-frequency scaling can be useful.

NoCs can enhance scalability in multi-core systems like Spike-MCryptCores by utilizing asynchronous Network-on-Chips for data flow. This approach, unlike MUX and DEMUX, can be limited by fan-out issues, making NoCs a promising future research project.

The system relies on offline training and cannot adapt due to the lack of an online training algorithm for multiple layer perceptron for SNN. Two solutions are using a dedicated CPU for data collection and training the SNN or using bio-plausible Spike-Timing Dependent Plasticity (STDP) online learning. STDP is a local learning approach with a single layer but can be used to adapt to different scenarios. By indicating the most firing neuron, the system can indicate possible control methods, potentially allowing for online adjustment. This could be one of the future works for our group to provide the ability to adjust the system online.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence

Table 5
Compare energy efficiency between algorithm models.

Work	Technique	Algorithm	Platform	Energy reduction (%)
Ababei et al. [50]	Dynamic voltage and frequency scaling (DVFS)	Distributed (DVFS) algorithm	CMOS-65nm (TSMC)	- Maximum 50.0 %
Zakaria et al. [51]	Dynamic voltage and frequency scaling (DVFS)	Programmable self-timed ring (PSTR)	CMOS-45nm (ST Microelectronics)	- Maximum 51.42 %
Chouchene et al. [52]	Clock Gating	Open Compute Project (OCP)	FPGA - Xilinx Virtex5	- Maximum 63.0 % - Minimum 37.0 %
Pande et al. [53]	Dynamic voltage and frequency scaling (DVFS)	Producer - Consumer FIFO	CMOS-90nm (TSMC)	- Maximum 32.2 %
Phan et al. [54]	Dynamic voltage and frequency scaling (DVFS)	Fuzzy logic algorithm	CMOS-65nm (TSMC)	- Maximum 43.0 %
Our work	Clock Gating	Spiking Neural Network (SNN) controller	CMOS-45nm	- Maximum 67.0 % - Minimum 39.0 % - Average 48.6 %

the work reported in this paper.

Data availability

No data was used for the research described in the article.

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.micpro.2024.105040.

References

[1] N. I. of S. and Technology, Advanced Encryption Standard (AES), U.S. Department of Commerce, Federal Information Processing Standard (FIPS) 197, 2001, https://doi.org/10.6028/NIST.FIPS.197.

[2] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, D.S. Nikolopoulos, Challenges and opportunities in edge computing, in: 2016 IEEE International Conference on Smart Cloud (SmartCloud), 2016, pp. 20–26, https://doi.org/10.1109/SmartCloud.2016.18.

[3] A. Botta, W. Donato, V. Persico, A. Pescapè, Integration of cloud computing and internet of things: a survey, Future Gener. Comput. Syst. 56 (2015), https://doi.org/10.1016/j.future.2015.09.021.

[4] J. Lichtman, H. Pfister, N. Shavit, The big data challenges of connectomics, Nat. Neurosci. 17 (2014) 1448–1454, https://doi.org/10.1038/nn.3837.

[5] Y. Xiao, Y. Jia, C. Liu, X. Cheng, J. Yu, W. Lv, Edge computing security: state of the art and challenges, in: Proc. IEEE, 2019, pp. 1–24, https://doi.org/10.1109/JPROC.2019.2918437.

[6] J.S. Khan, J. Ahmad, S. Ahmed, H. Siddiqa, S. Abbasi, S. Kayhan, DNA key based visual chaotic image encryption, J. Intell. Fuzzy Syst. (2019) 1–13, https://doi.org/10.3233/JIFS-182778.

[7] *Intelligent Computing* S.F. Abbasi, J. Ahmad, J.S. Khan, M.A. Khan, S.A. Sheikh, Visual meaningful encryption scheme using intertwinning logistic map, in: K. Arai, S. Kapoor, R. Bhatia (Eds.), Advances in Intelligent Systems and Computing, Springer International Publishing, Cham, 2019, pp. 764–773, https://doi.org/10.1007/978-3-030-01177-2_56.

[8] J.S. Khan, J. Ahmad, S.F. Abbasi, Arshad, S.K. Kayhan, DNA sequence based medical image encryption scheme. 2018 10th Computer Science and Electronic Engineering (CEEC), 2018, pp. 24–29, https://doi.org/10.1109/CEEC.2018.8674221.

[9] A. Soltani, S. Sharifian, An ultra-high throughput and fully pipelined implementation of AES algorithm on FPGA, Microprocess. Microsyst. 39 (7) (2015) 480–493, https://doi.org/10.1016/j.micpro.2015.07.005.

[10] K. Rahimunnisa, P. Karthigaikumar, N. Christy, S. Kumar, J. Jayakumar, PSP: Parallel sub-pipelined architecture for high throughput AES on FPGA and ASIC, Open Comput. Sci. 3 (4) (2013) 173–186, https://doi.org/10.2478/s13537-013-0112-2.

[11] P. Liu, J. Hsiao, H. Chang, C. Lee, A 2.97 Gb/s DPA-resistant AES engine with self-generated random sequence, in: 2011 Proceedings of the ESSCIRC (ESSCIRC), 2011, pp. 71–74, https://doi.org/10.1109/ESSCIRC.2011.6044917.

[12] K. Gaj, P. Chodowiec, FPGA and ASIC implementations of AES. Cryptographic engineering, Springer, 2009, pp. 235–294.

[13] D.-H. Bui, D. Puschini, S. Bacles-Min, E. Beigne, X.-T. Tran, AES datapath optimization strategies for low-power low-energy multisecurity-level internet-of-things applications, IEEE Trans. Very Large Scale Integr. VLSI Syst. 25 (12) (2017) 3281–3290, https://doi.org/10.1109/TVLSI.2017.2716386.

[14] A.A. Pammu, W. Ho, N.K.Z. Lwin, K. Chong, B. Gwee, A high throughput and secure authentication-encryption aes-ccm algorithm on asynchronous multicore processor, IEEE Trans. Inf. Forensics Secur. 14 (4) (2019) 1023–1036, https://doi.org/10.1109/TIFS.2018.2869344.

[15] S. Davidson, S.B. Furber, Comparison of artificial and spiking neural networks on digital hardware, Front. Neurosci. 15 (2021). Accessed: 2022. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2021.651141.

[16] S. Furber, Large-scale neuromorphic computing systems, J. Neural Eng. 13 (Aug. 2016), https://doi.org/10.1088/1741-2560/13/5/051001.

[17] C. Lee, S.S. Sarwar, P. Panda, G. Srinivasan, K. Roy, Enabling spike-based backpropagation for training deep neural network architectures, Front. Neurosci. 14 (2020). Accessed: 2022. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnins.2020.00119.

[18] Y. Wu, L. Deng, G. Li, J. Zhu, Y. Xie, L.P. Shi, Direct training for spiking neural networks: faster, larger, better, Proc. AAAI Conf. Artif. Intell. 33 (2019) 1311–1318, https://doi.org/10.1609/aaai.v33i01.33011311.

[19] W. Zhao, Y. Ha, M. Alioto, AES architectures for minimum-energy operation and silicon demonstration in 65nm with lowest energy per encryption, in: 2015 IEEE International Symposium on Circuits and Systems (ISCAS), 2015, pp. 2349–2352, https://doi.org/10.1109/ISCAS.2015.7169155.

[20] V.- Hoang, V.- Dao, C.- Pham, Design of ultra-low power AES encryption cores with silicon demonstration in SOTB CMOS process, Electron. Lett. 53 (23) (2017) 1512–1514, https://doi.org/10.1049/el.2017.2151.

[21] P. Maene and I. Verbauwhede, “Single-cycle implementations of block ciphers,” Jan. 2016, pp. 131–147. 10.1007/978-3-319-29078-2_8.

[22] S.K. Mathew, et al., 53 Gbps Native GF$(2^4)^2$ composite-field AES-encrypt/decrypt accelerator for content-protection in 45 nm high-performance microprocessors, IEEE J. Solid-State Circuits 46 (4) (2011) 767–776, https://doi.org/10.1109/JSSC.2011.2108131.

[23] B. Buhrow, K. Fritz, B. Gilbert, E. Daniel, A highly parallel AES-GCM core for authenticated encryption of 400 Gb/s network protocols, in: 2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig), 2015, pp. 1–7, https://doi.org/10.1109/ReConFig.2015.7393321.

[24] L. Henzen, W. Fichtner, FPGA parallel-pipelined AES-GCM core for 100G Ethernet applications, in: 2010 Proceedings of ESSCIRC, 2010, pp. 202–205, https://doi.org/10.1109/ESSCIRC.2010.5619894.

[25] EDN, “Reducing IC power consumption: Low-power design techniques,” EDN. Accessed: Feb. 16, 2022. [Online]. Available: https://www.edn.com/reducing-ic-power-consumption-low-power-design-techniques/.

[26] H. Mahmoodi, V. Tirumalashetty, M. Cooke, K. Roy, Ultra low-power clocking scheme using energy recovery and clock gating, IEEE Trans. Very Large Scale Integr. VLSI Syst. 17 (1) (2009) 33–44, https://doi.org/10.1109/TVLSI.2008.2008453.

[27] S. Wimer, A. Albahari, A look-ahead clock gating based on auto-gated flip-flops, IEEE Trans. Circuits Syst. Regul. Pap. 61 (5) (2014) 1465–1472, https://doi.org/10.1109/TCSI.2013.2289404.

[28] H. Qiao, J. Chen, X. Huang, A survey of brain-inspired intelligent robots: integration of vision, decision, motion control, and musculoskeletal systems, IEEE Trans. Cybern. (2021) 1–14, https://doi.org/10.1109/TCYB.2021.3071312.

[29] F.M.M. ul Islam, M. Lin, Hybrid DVFS scheduling for real-time systems based on reinforcement learning, IEEE Syst. J. 11 (2) (2017) 931–940, https://doi.org/10.1109/JSYST.2015.2446205.

[30] D. Liu, S.-G. Yang, Z. He, M. Zhao, W. Liu, CARTAD:compiler-assisted reinforcement learning for thermal-aware task scheduling and DVFS on multicores, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. (2021) 1, https://doi.org/10.1109/TCAD.2021.3095028. –1.

[31] H. Jung, M. Pedram, Supervised learning based power management for multicore processors, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 29 (9) (2010) 1395–1408, https://doi.org/10.1109/TCAD.2010.2059270.

[32] L. Deng, et al., Rethinking the performance comparison between SNNS and ANNS, Neural Netw 121 (2020) 294–307, https://doi.org/10.1016/j.neunet.2019.09.005. Tháng Một.

[33] C. Frenkel, M. Lefebvre, J.-D. Legat, D. Bol, A 0.086-mm$ˆ2$ 12.7-pJ/SOP 64k-Synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm CMOS, IEEE Trans. Biomed. Circuits Syst. 13 (1) (2019) 145–158, https://doi.org/10.1109/TBCAS.2018.2880425.

[34] C. Frenkel, J.-D. Legat, D. Bol, MorphIC: A 65-nm 738k-Synapse/mm^2 quad-core binary-weight digital neuromorphic processor with stochastic spike-driven online learning, IEEE Trans. Biomed. Circuits Syst. 13 (5) (2019) 999–1010, https://doi.org/10.1109/TBCAS.2019.2928793.
[35] D. Khodagholy, et al., NeuroGrid: recording action potentials from the surface of the brain, Nat. Neurosci. 18 (2) (2015) 2, https://doi.org/10.1038/nn.3905. Art. no.
[36] S. Schmitt, et al., Neuromorphic hardware in the loop: Training a deep spiking network on the BrainScaleS wafer-scale system, in: 2017 International Joint Conference on Neural Networks (IJCNN), 2017, pp. 2227–2234, https://doi.org/10.1109/IJCNN.2017.7966125.
[37] A. Ben Abdallah, K.N. Dang, Toward robust cognitive 3D brain-inspired cross-paradigm system, Front. Neurosci. 15 (2021). Accessed: Feb. 16, 2022. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2021.690208.
[38] F. Akopyan, et al., TrueNorth: design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip, Comput.-Aided Des. Integr. Circuits Syst. IEEE Trans. On 34 (2015) 1537–1557, https://doi.org/10.1109/TCAD.2015.2474396.
[39] M. Davies, et al., Advancing neuromorphic computing with Loihi: a survey of results and outlook, in: Proc. IEEE 109, 2021, pp. 911–934, https://doi.org/10.1109/JPROC.2021.3067593.
[40] T. Hwu, J. Isbell, N. Oros, J. Krichmar, A self-driving robot using deep convolutional neural networks on neuromorphic hardware, in: 2017 International Joint Conference on Neural Networks (IJCNN), 2017, pp. 635–641, https://doi.org/10.1109/IJCNN.2017.7965912.
[41] T. Hwu, A.Y. Wang, N. Oros, J.L. Krichmar, Adaptive robot path planning using a spiking neuron algorithm with axonal delays, IEEE Trans. Cogn. Dev. Syst. 10 (2) (2018) 126–137, https://doi.org/10.1109/TCDS.2017.2655539.
[42] K.D. Fischl, et al., Neuromorphic self-driving robot with retinomorphic vision and spike-based processing/closed-loop control, in: 2017 51st Annual Conference on Information Sciences and Systems (CISS), 2017, pp. 1–6, https://doi.org/10.1109/CISS.2017.7926179.
[43] P.U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, M. Pfeiffer, Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing, in: 2015 International Joint Conference on Neural Networks (IJCNN), 2015, pp. 1–8, https://doi.org/10.1109/IJCNN.2015.7280696.
[44] P. Diehl, M. Cook, Unsupervised learning of digit recognition using spike-timing-dependent plasticity, Front. Comput. Neurosci. 9 (2015). Accessed: Feb. 17, 2022. [Online]. Available: https://www.frontiersin.org/article/10.3389/fncom.2015.00099.
[45] N. Kasabov, Evolving spiking neural networks and neurogenetic systems for spatio- and spectro-temporal data modelling and pattern recognition, in: J. Liu, C. Alippi, B. Bouchon-Meunier, G.W. Greenwood, H.A. Abbass (Eds.), Advances in Computational Intelligence: IEEE World Congress on Computational Intelligence, WCCI 2012, Brisbane, Australia, June 10-15, 2012. Plenary/Invited Lectures, Springer, Berlin, Heidelberg, 2012, pp. 234–260, https://doi.org/10.1007/978-3-642-30687-7_12. Lecture Notes in Computer Science.
[46] J.H. Lee, T. Delbruck, M. Pfeiffer, Training deep spiking neural networks using backpropagation, Front. Neurosci. 10 (2016). Accessed: Feb. 17, 2022. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2016.00508.
[47] M. Pfeiffer, T. Pfeil, Deep learning with spiking neurons: opportunities and challenges, Front. Neurosci. 12 (2018) 1–13.
[48] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, S.-C. Liu, Conversion of continuous-valued deep networks to efficient event-driven networks for image classification, Front. Neurosci. 11 (2017) 1–12.
[49] Fang, Wei and Chen, Yanqi and Ding, Jianhao and Chen, Ding and Yu, Zhaofei and Zhou, Huihui and Tian, Yonghong and other contributors, "SpikingJelly." Feb. 15, 2022. Accessed: Feb. 16, 2022. [Online]. Available: https://github.com/fangwei123456/spikingjelly.
[50] C. Ababei, N. Mastronarde, Benefits and costs of prediction based DVFS for NoCs at router level, in: 2014 27th IEEE International System-on-Chip Conference (SOCC), 2014, pp. 255–260, https://doi.org/10.1109/SOCC.2014.6948937.
[51] H. Zakaria, L. Fesquet, Process variability robust energy-efficient control for nano-scaled complex SoCs. 10th Edition of Faible Tension Faible Consommation (FTFC'11), IEEE Computer Society, Marrakech, Morocco, 2011, pp. 95–98, https://doi.org/10.1109/FTFC.2011.5948928.
[52] W. Chouchene, A. Brahim, A. Zitouni, N. Abid, R. Tourki, A low power network interface for network on chip, in: Int. Multi-Conf. Syst. Signals Devices SSD11 - Summ. Proc., 2011, https://doi.org/10.1109/SSD.2011.5767464.
[53] P. Pande, C. Grecu, M. Jones, A. Ivanov, R. Saleh, Performance evaluation and design trade-offs for network-on-chip interconnect architectures, Comput. IEEE Trans. On 54 (2005) 1025–1040, https://doi.org/10.1109/TC.2005.134.
[54] H.-P. Phan, X.-T. Tran, T. Yoneda, Power consumption estimation using VNOC2.0 simulator for a fuzzy-logic based low power Network-on-Chip, in: 2017 IEEE International Conference on IC Design and Technology (ICICDT), 2017, pp. 1–4, https://doi.org/10.1109/ICICDT.2017.7993515.
[55] K.N. Dang, M. Meyer, Y. Okuyama, A.B. Abdallah, A low-overhead soft–hard fault-tolerant architecture, design and management scheme for reliable high-performance many-core 3D-NoC systems, J. Supercomput. 73 (6) (2017) 2705–2729, https://doi.org/10.1007/s11227-016-1951-0.

PHAM-KHOI DONG received a Ph.D. degree at VNU University of Engineering and Technology. He received his Bachelor degree in Electronics and Telecommunications from Le Quy Don Technical University and Master degree in Electronics and Telecommunications from VNU University of Engineering and Technology. His research interest includes system-on-chip design, hardware accelerator for cryptography.

KHANH N. DANG is currently an associate professor of The University of Aizu, Japan. He received his B.Sc., M.Sc., and Ph. D. degree from Vietnam National University Hanoi (VNU), University of Paris-XI, and The University of Aizu, Japan, in 2011, 2014, and 2017, respectively. He has served as a TPC co-chair of several IEEE conferences such as MCSoC 2019/2021 and APCCAS 2020. His research interests include System-on Chips/Network-on-Chips, 3D-ICs, neuromorphic computing, and fault-tolerant system. Dr. Dang has published more than 30 peer-reviewed publications and 4 Japanese patent applications. His also co-authors an incoming book titled "Neuromorphic Computing Principles and Organization" on Springer published in 2022.

DUY-ANH NGUYEN is currently a researcher at the Information Technology Institute, Vietnam National University, Hanoi. He was a Ph.D. student at VNU University of Engineering and Technology and Joint Technology Innovation and Research Centre between Vietnam National University Hanoi (VNU) and the University of Technology Sydney. He received his Bachelor degree from Nanyang Technological University, Singapore and Master degree from Southwest Jiao Tong University, China. His research interest includes system-on-chip design, hardware accelerator for artificial intelligent.

XUAN-TU TRAN received a Ph.D. degree in 2008 from Grenoble INP (at the CEA-LETI), France, in Micro Nano Electronics. He is currently a full professor at Vietnam National University, Hanoi (VNU), and the Director of VNU Information Technology Institute. He was an invited professor at the University Paris-Sud 11, France (2009, 2010, and 2015), University of Electro-Communication, Tokyo (2019), Grenoble INP (2011), and adjunct professor at University of Technology Sydney (2017–2023). He was Director for the VNU Key Laboratory for Smart Integrated Systems (SISLAB) from 2016 to 2021. His research interests include design and test of systems-on-chips, networks-on- chips, design-for-testability, asynchronous/synchronous VLSI design, low power techniques, and hardware architectures for multimedia applications. He has published 3 books, 4 patents and more than 120 peer-reviewed publications in these areas. He is a Senior Member of the IEEE, IEEE Circuits and Systems (CAS), IEEE Solid-State Circuits and Systems (SSCS), member of IEICE, and the Executive Board of the Radio Electronics Association of Vietnam (REV).