SPLEAT: SPiking Low-power Event-based ArchiTecture for in-orbit processing of satellite imagery

Conference Paper · July 2022

DOI: 10.1109/IJCNN55064.2022.9892277

4 authors, including:

Nassim Abderrahmane
Menta S.A.S, Sophia Antipolis, France
10 PUBLICATIONS **133** CITATIONS

SEE PROFILE

Benoit Miramond
Université Côte d'Azur
105 PUBLICATIONS **757** CITATIONS

SEE PROFILE

SPLEAT: SPiking Low-power Event-based ArchiTecture for in-orbit processing of satellite imagery

1st Nassim Abderrahmane*
IRT Saint Exupéry
Sophia-Antipolis, France

2nd Benoît Miramond
Université Côte d'Azur, CNRS, LEAT
Sophia-Antipolis, France

3rd Erwann Kervennic
IRT Saint Exupéry
Sophia-Antipolis, France

4th Adrien Girard
IRT Saint Exupéry
Sophia-Antipolis, France

***Abstract*—In this paper, we present SPLEAT, a SPiking Low-power Event-based ArchiTecture for the hardware deployment of Spiking Neural Networks (SNNs). SPLEAT has a configurable architecture that allows integrating several hardware modules representing different neural layers and, therefore, gives the ability to deploy a wide range of spiking convolutional neural networks. Thanks to its event-based structure, SPLEAT takes advantage of the asynchronous behavior of spiking data to realize prediction efficiently. SPLEAT has been synthesized on a Cyclone V Field-Programmable Gate Array (FPGA) embedded on OPS-SAT, a 3U nano-satellite launched on December 18, 2019 by the European Space Agency (ESA). SPLEAT has been used to perform in-orbit binary cloud classification on images provided by a 50 meters resolution sensor, which is still operational to this date. A second goal of the in-orbit experiment was to confront deep learning and bio-inspired neural networks. Consequently, in addition to SNNs executed using SPLEAT, classical Convolutional Neural Networks (CNNs) have also been deployed and evaluated on the same hardware target embedded on the OPS-SAT satellite. The comparison between these neural families reveal a notable gain in terms of resources occupation for SPLEAT, reducing it by a factor of 6.62 while having equivalent classification performances and an FPGA power consumption reduced by a factor of 5.91. To our best knowledge, this in-orbit deployment of SNNs constitutes a world premiere in the fields of bio-inspired neural networks and aerospace.**

***Index Terms*—Spiking Neural Networks, Satellite Imaging, Neuromorphic Computing, Hardware Architecture, FPGA, Artificial Intelligence, Deep Learning, Nano-satellite, Binary cloud classification, Embedded Systems**

I. Introduction

Spiking neural networks (SNNs) offer opportunities to efficiently implement Artificial Intelligence (AI) applications on embedded systems. They are a model of neural generation attempting to mimic the function and structure of some biological neural regions. This bio-inspired approach allows for simpler computations than classical Artificial Neural Networks (ANNs). In addition to their computational architecture which is well suited for efficient hardware implementation, recent advances in their learning phase have rendered their performance close to the one of the formal neural networks [1]. Consequently, SNNs constitute a very promising alternative to formal networks for use in embedded AI application systems. Indeed, spiking neural networks comprise bio-inspired neurons that use action potentials, called spikes, to perform their computations. A spiking neuron, such as the Integrate-and-Fire model, responds to an input stimulus by integrating a synaptic weight that is associated with its connection to the emitting neuron [2]. This process alters the internal potential so that, if there is sufficient stimulation, the neuron emits an output spike [3]. This simple operation is convenient for a hardware implementation. Moreover, this operation is known to be spatio-temporal, since it refers to the spatial location of the neuron in the network and the dependence of its computations on time. This spatio-temporal property of SNNs is well suited to event-driven architectures. Indeed, these neurons remain idle when there is no event to process, making SNNs inherently event-driven models. Furthermore, as shown [4], the spiking data is spatio-temporally sparse and, in the case of Deep Neural Networks (DNNs), the amount of spikes decreases drastically as one goes deeper in the network layers. This property offers the possibility of adopting different computation parallelism and to adapt the architecture to the data. In this context, a hybrid architecture is proposed in [5], in which: the first layer, which generates a large amount of spikes, is implemented in a parallel manner; and the last layers, which generate few spikes, are time multiplexed. For scalability reasons and to deal with spiking DNNs, it is necessary to adopt multiplexed implementations. Fortunately, the reduction in the number of operations with spiking DNN layers will mitigate the latency increase caused by such multiplexing. Therefore, due to these aspects, SNNs represent a promising solution for the deployment of AI applications on embedded systems.

Related work: There are several hardware targets for deploying ANNs for embedded AI applications. First, the standard Central Processing Unit (CPU) is one of these types and was considered in [6]. However, except when using large computational infrastructures, CPU-based designs result in extended execution time and hence make them unsuitable for time-constrained embedded applications. To accelerate the processing, the use of Graphical Processing Units (GPUs) is the most popular alternative [7], [8]. However, this is not the optimal solution due to their high power consumption. Consequently, instead of using these general purpose systems, dedicated ANN hardware architectures, known as neuromorphic hardware systems, represent a strong alternative.

*Corresponding author e-mail :nassim.abderrahmane@irt-saintexupery.com

To design such systems, both digital and analog electronic substrates can be considered. The most widely used substrates for this neuromorphic design are digital systems. The reason behind that is the fact that they benefit from the maturity of their manufacturing technologies and programming facilities. Another reason is that, as shown in [9], the implementation of ANNs below $22nm$ technology is more efficient on digital systems than analog when considering chip area and scalability. Digital systems can be implemented using a Field Programmable Gate Array (FPGA) or an Application Specific Integrated Circuit (ASIC).

There are some recent spiking neuromorphic hardware systems proposed in the literature and which consist of ASIC/FPGA chips, which are dedicated for deploying SNNs. Some of them, such as SpiNNaker, TrueNorth [10] or Loihi [11], are ASIC systems built to emulate and simulate very large spiking networks which may be used in neuroscience studies and experiments. There are others, such as ConvNode [12], DYNAPS [13], which are FPGA-based digital systems dedicated to deploying NNS for embedded AI applications. A review of these neuromorphic systems is proposed in [5].

In this paper, we present SPLEAT, a SPiking Low-power Event-based ArchiTecture (SPLEAT) dedicated for deploying SNNs on FPGA. Equipped with a modular and time-multiplexed structure, SPLEAT allows integrating a variable number of SNN layer modules allowing the support of a wide range of deep and convolutional SNN topologies. In addition, it has an event-based structure that fits conveniently with the spiking data behavior. This architecture has been deployed on a Cyclone V FPGA embedded on a 3U nano-satellite ($30 \times 10 \times 10$ cm size), namely OPS-SAT, which was launched on December 18, 2019 by the European Space Agency (ESA). It is used to perform in-orbit cloud segmentation on images provided by an embedded camera with a 50 meters ground sampling distance [14]. In addition, SPLEAT has been compared to formal CNNs on the same target. The results show a notable gain in terms of logic resources usage for SPLEAT with a reduction by a factor of 6.62 while having equivalent classification performances. To our best knowledge, this experiment of SNN deployment on an in-orbit satellite is a world premiere in the bio-inspired neural networks and aerospace domains.

In the remainder of the paper, we first describe SNNs briefly with a review of their structure, data representation and their associated training techniques. Second, we present the SPLEAT architecture, in section III. Afterwards, in section IV, the deployment methodology of SPLEAT architecture in the in-orbit satellite, OPS-SAT, is described. Then, in the section V, the deployment results of SPLEAT are reported and discussed. Finally, a conclusion of the paper is provided in section VI.

II. Spiking Neural Networks

A number of neural coding schemes have been proposed in neuroscience literature, including rate, temporal, and rank-order encoding. This encoding scheme has a significant effect on the energy efficiency because it influences the amount of spiking events and operations during inference. The most frequently employed approaches range from rate to temporal paradigms.

Rate coding: Rate coding is the most widespread method for converting data to the spiking domain. Indeed, spike trains with frequencies proportional to data intensity are obtained from the raw data with this neural coding conversion scheme. Some rate coding approaches add a white noise to the generated periodic spike train to improve robustness to noise and the model's performance [15].

Time coding: With temporal coding, data is encoded in the spiking domain by converting it to spike emission times. This coding approach allows the use of few spikes for representing the input data [16].

A. Spiking neurons

Several spiking neuron models influence highly neuromorphic architecture efficiency in terms of accuracy and hardware cost. In this context, the most computationally simplest models are the Leaky and non-leaky Integrate-and-Fire (L)IF neuron. Moreover, IF neurons are known to be sufficient for most classification applications, especially with static data [17], [18]. In this case, SNNs with IF neurons obtain higher results than SNNs based on the other spiking neurons such as the Izhikevich model [19]. When compared to the formal Artificial Neural Network (ANN) neuron, they are less resource-intensive because the spiking neuron involves only addition and comparison whereas multiplicative operations and a nonlinear function are used with perceptrons [2].

B. Training methods

In addition to the neuron and coding models, the training algorithm is another aspect that influences the spiking hardware architecture efficiency. There are two possible approaches to train SNNs, supervised and unsupervised learning. With supervised learning, labeled datasets are used to learn the SNN by adapting its parameters to the data. With unsupervised learning, based on the Hebbian rule the synaptic weights are updated to adapt them with unlabeled data [20]. The supervised learning approach achieves much higher performances and is more mature. There are two possible ways to realize such supervised training: ANN-SNN conversion and recently, direct spiking back-propagation. The ANN-SNN conversion approach is addressed by several works, such as in [3], and is considered as more mature. Indeed, the ANN is trained using a learning approach that benefits from the maturity of machine learning and deep learning fields. Recently, the spike based learning approaches have become more competitive by performing as well as ANNs while reducing the amount of spiking events during inferences. Moreover, a recent work [1] has shown that this learning model reduces inference latency, where SNNs result in a $10\times$ lower execution time than those of SNNs learned with ANN-SNN conversion. However, spike-based learning requires a higher number of learning iterations and a dedicated training framework compared to ANN-SNN conversion approach.

III. SPLEAT ARCHITECTURE

SPLEAT is an event-based hardware architecture dedicated for SNN inference as an embedded AI application on FPGA platforms. The architecture is generic due to a modular structure that allows the integration of a variable number of modules, which represent the different layers of an SNN. In the work [5], it has been shown that parallel architectural design is not adapted to DNN hardware implementation because of leading to significant hardware footprints in terms of chip surface. Moreover, it has also been shown that multiplexing hardware is more efficient when dealing with DNNs deployment on limited resource targets such as embedded systems [10], [21]. Therefore, to make SPLEAT scalable and fit with spiking DNNs, a time-multiplexing design is adopted. Besides the time-multiplexed and modular structure of SPLEAT, which makes it generic and scalable, it employs an event-based communication and computation system that handles the spiking data efficiently. In the following subsections, the architecture will be described through a presentation of its computing and interfacing mechanisms and its different components.

A. General overview

The architecture comprises Neural Processing Modules (NPMs) that have two different modes (convolution and dense), a Spike Generation Module (SGM) and a Terminate Delta Module (TDM). Figure 1 shows an overview of SPLEAT, where each layer of the SNN is represented by a convolution or a dense NPM, according to its type. Moreover, it comprises an input SGM that converts data to spiking domain and which is placed before the convolution NPMs. Finally, an output TDM is located at the last stage of SPLEAT for which the role is enacting the classification and ending the inference process. Due to this structure, with an NPM per layer operating concurrently, a pipeline computation is ensured, which allows reducing execution time. Moreover, the temporal behavior of spiking data combined with a pipeline architecture reduces the execution time, because, if a same output neuron fires several times in a row, the TDM orders to stop the processing before having completed processing all the input data.

1) Event-based processing: SPLEAT modules are interconnected using interfaces similar to First-In First-Out (FIFO) ports. This interface facilitates the interconnection of multiple modules and allows an event-based computation that is appropriate to the spiking data behavior. The spiking events give information about the location of their emitting neurons in the network. Due to a feed-forward structure, the layer identifier is implicit and, hence, these events only indicate their position inside the layer. Assigning a timestamp to these events is not adopted for several reasons. First, each NPM has a FIFO memory that stores in an ordered way the output spikes of the represented layer according to their emission time. Second, at the network level, spikes are sorted correctly because they are stored in FIFOs connected in a feed-forward manner. Finally, the events do not contain timing information because the IF neurons do not use *"time"* in their computations. Thus, just respecting the order of their emissions is sufficient. This computing mechanism, based on ordering spikes, also known as rank order coding is described in [22].

2) Communication protocol: Figure 2 shows the I/O interface of an NPM module; it consists of three inputs and three outputs connecting the NPM to its previous and next layers. First, the NPM checks for the presence of input events using the *"i_Empty"* port. When an event is present, this *"i_Empty"* is equal to *"0"* and, consequently, when it is equal to *"1"*, the module remains in idle state. In case of presence of input event, it is considered for processing by reading its content via the *"i_Event"* port. Once the NPM module has read the event, it sets "o_Rd-Event" to high (equal to *"1"*) to inform the previous layer. Then, if the processing of the event produces an output event, and if the FIFO is empty, *"o_Empty"* is reset to low. Then, the spike is recorded in the FIFO and can be read by the next layer's NPM via the *"o_Event"* port. When this next layer's NPM reads an event from the FIFO, *"i_Rd_Event"* is set high, which removes the first event in the FIFO queue. If that event is the last, then *"i_Empty"* is set high. Notice that the FIFO *"Full"* signal is internal to the NPM and used to control reading and writing of input/output events.

B. Spike Generation Module

In case of non-event-based sensors, the input raw data received by an SNN architecture must be transformed to spiking domain before its processing. SPLEAT comprises the SGM module that performs a neural data conversion based on rate coding. The SGM converts image pixels to spiking events emitted in times proportional to the pixel intensities.

The rate-based encoding function is implemented as a simple Look-Up-Table (LUT) giving the result of its application to the pixel directly, these are computed offline using a python script. This technique allows reducing the hardware cost. Within this module, a counter is used to represent the virtual time in function of the FPGA cycles, a counter identifying the channel of the pixel (R, G and B) and a last one giving the address of the pixel inside the image (0 to 28×28 in our experiments). A FIFO buffer is used to memorize the addresses of the output spikes as events. Spike emission times are retrieved from the LUT using the addresses of the input pixels. Whenever one of these spike emission times is equal to the virtual time given by the first counter, an event with the address of the pixel is registered in the FIFO. In this work, we have designed two versions of the SGM: time-multiplexed and channel-level parallel.

First, with the time multiplexed version, the pixels of a $w * h$ input image pixels are processed sequentially spending $3 * w * h$ FPGA cycles to generate the spikes that belong to one virtual time. Second, in the channel-level parallel version of this SGM, the RGB channels of the input image pixels are processed concurrently. Therefore, generating the spikes belonging to a single virtual time is done in $w*h$ cycles, which is $3\times$ faster than the first SGM version. In the remaining, the two versions are respectively referred to as SNN^1 and SNN^2.

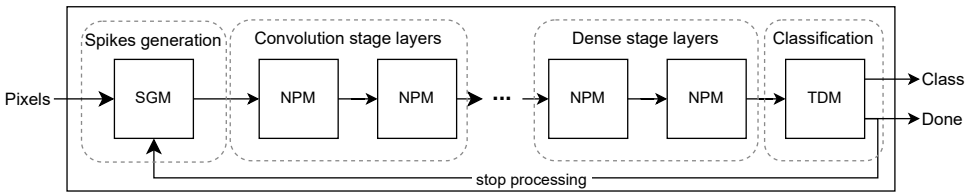

Fig. 1: Schematic diagram of SPLEAT architecture.

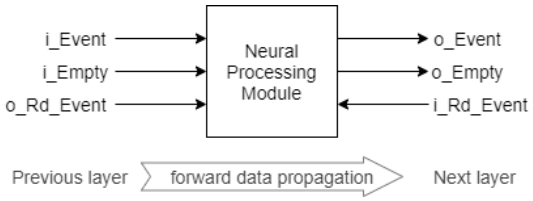

Fig. 2: NPM's I/O interface representation.

C. Neural Processing Module

In this section, we describe the Neural Processing Module shown in figure 3. The NPM is composed of two parts, a *Control_Module* and a *Computing_Module*. The control and computing modules are connected between each other to communicate neurons and weights addresses, enable computations and inform the state of the FIFO buffer. On one hand, the *Control_Module* manages the NPM state by controlling its state and enabling the *Computing_Module*. On the other hand, the computing core is dedicated for processing the logical neurons of a single layer.

1) Control Module: The *Control_Module* internal structure, illustrated in figure 4, shows that it is composed of two sub-modules: a *Finite_State_Machine* and an *Address_Manager* module. The role of this control unit is to monitor the NPM components and ensure a coherent operation and communication with its neighboring NPMs, i.e. reading input events from the previous layer, process them in a time-multiplexed way and then communicate the generated output events to the next layer.

***Finite_State_Machine*:** The FSM role is to change the state

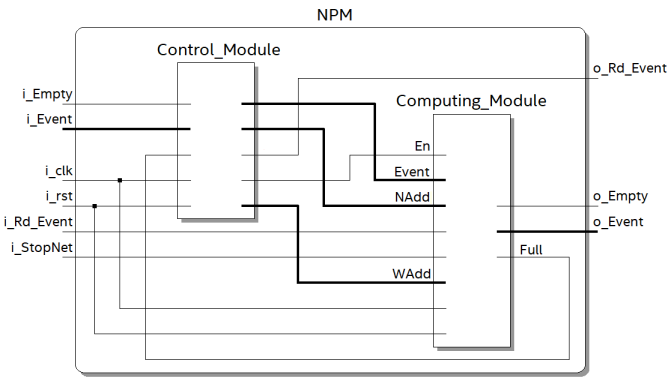

Fig. 3: NPM structure.

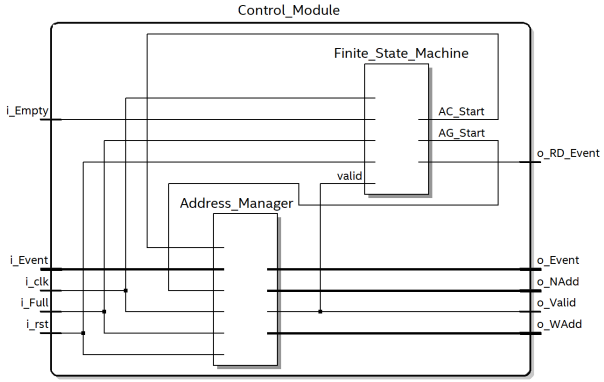

Fig. 4: *Control_Module*'s internal structure.

of the control sub-modules and to enable the computing core. A state diagram of this FSM is shown in figure 5.

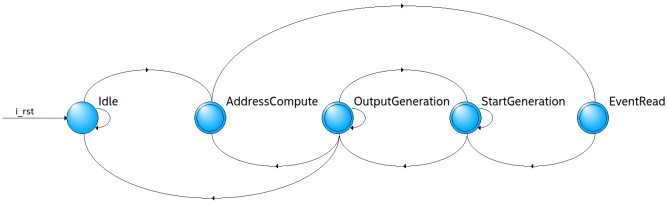

Fig. 5: NPM Finite State Machine.

The FSM is composed of 5 states: *"Idle"*, *"AddressCompute"*, *"StartGeneration"*, *"EventRead"* and *"OutputGeneration"*. At reset, the FSM is in "Idle" state where all the outputs are reset. When an input event occurs, the FSM goes to *"AddressCompute"* state to determine the neuron and weight addresses. Next to this, the FSM transits to *"StartGeneration"*, where it disables *Address_Compute* module and enables the *Address_Generator* module. In case the FIFO is not full, the state machine transits to *"OutputGeneration"*, where the neuron and weight addresses are communicated to the *Computing_Module*. When this address generation finishes, the FSM passes to *"EventRead"* state, where the event reading action is signaled to the previous layer. This process is repeated until the classification ends.

***Address_Manager*:** As shown in figure 6, it is composed of an *Address_Compute* and an *Address_Generator*. Its role is to determine the synaptic weights and neurons needed by the *Computing_Module* to process the input spiking event.

***Address_Compute*:** This module is a hardware implementa-

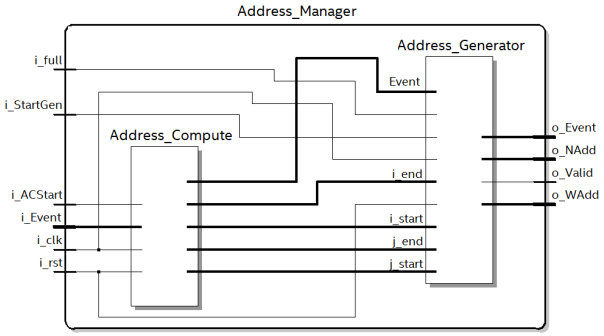

Fig. 6: *Address_Manager*'s internal structure.

tion of the algorithm 1, which determines the required neuron and weight addresses to sequentially process the input event by a convolution layer's NPM. Its role is reducing the number of hardware connections and multiplexing the processing. To this end, the module maps the input event into (E_C, E_H, E_W) format which gives its exact spatial position in the input feature map. Then, it uses this information to determine the coordinates range of the neurons that intersect the input event in terms of width and height coordinates start and end indices.

Map $Event$ to CHW coordinates:
$(E_C, E_H, E_W) = Map(Event)$
Neuron's width coordinates:
Result: (i_{start}, i_{end})
if $E_W - F + 1 < 0$ **then**
  $i_{start} = 0$;
  $i_{end} = \lfloor E_W/S \rfloor$;
else if $E_W + F - 1 >= N_{in} - 1$ **then**
  $i_{start} = \lfloor (E_W - F)/S \rfloor + 1$;
  $i_{end} = N_{out} - 1$;
else
  $i_{start} = \lfloor (E_W - F)/S \rfloor + 1$;
  $i_{end} = \lfloor E_W/S \rfloor$;
end

Algorithm 1: Convolution *Address_Compute* algorithm

With (E_C, E_H, E_W) the input event's channel, height and width coordinates. (i_{start} , i_{end}) the start and end indices of the input event's intersecting convolution neurons width coordinates range. F the kernel's width, S the stride, N_{in} / N_{out} the input / output Feature Map (FM) and $\lfloor ./. \rfloor$ represents integer division. The same methodology can be applied to get the height coordinates range (j_{start} , j_{end}). A thorough description of this algorithm is given in [23].

***Address_Generator*:** This module, shown in figure 7, represents the hardware implementation of the algorithm 2. The algorithm describes the methodology to generate the neuron and weight addresses (@Neuron, @Weight) by using the start/end indices given by the *Address_Compute* module. The nested loops iterating over width (i), height (j) and kernel (k) coordinates, present in the algorithm 2, are unrolled and implemented as three superposed counters. These counters give the width ($NAdd_H$), height ($NAdd_W$) and kernel (K) coordinates. Those are then used to determine the neuron and weight addresses (*@Neuron* and *@Weight*). Then, these addresses are used by the *Computing_Module* to retrieve from memory the neuron's internal potential and synaptic weight. In

for $i = i_{start};\ i \leq i_{end};\ i++$ **do**
  for $j = j_{start};\ j \leq j_{end};\ j++$ **do**
    for $k = 0;\ k \leq N_{Kernel} - 1;\ k++$ **do**
      $@Neuron = i + j * N_{out} + k * N_{out}^2$;
      $WAdd_W = E_W - i * S$;
      $WAdd_H = E_H - j * S$;
      $WAdd_C = E_C$;
      $@Weight = WAdd_W + WAdd_H * F + WAdd_C * F^2 + k * N_{Channel} * F^2$;
    end
  end
end

Algorithm 2: Address generation algorithm

the algorithm 2, N_{Kernel} is the number of convolution kernels (filters) and $N_{Channel}$ to the number of channels (number of input FMs). A detailed explanation of this algorithm is given in [23].

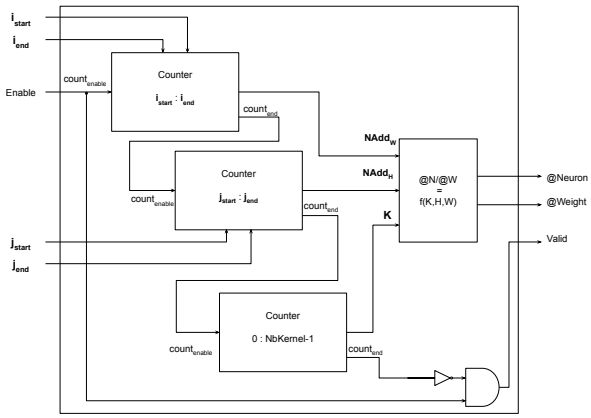

Fig. 7: *Address_Generator*'s internal structure.

2) Computing Module: The second part of the NPM is its *Computing_Module*, which is responsible for the integration and generation of spiking events. As shown in figure 8, the *Computing_Module* is composed of two memory blocks, an integrate-and-fire neuron and a FIFO buffer. One of the memory blocks is a Read Only Memory (ROM) that is used for saving the synaptic weights and the other is a Random Access Memory (RAM) that is used to read and write the neuron's activities. The IF neuron block role is processing the input events of the layer. The FIFO is there to store the output spikes. The computing module is connected to the control module and to the next layer's NPM.

Functioning: When the control module enables the computing module, i.e. *"i_En"* is high, the data present on the input ports are valid. First, it accesses the RAM to retrieve the neuron activity using the address port *"i_NAdd"*. At the same

clock cycle, *"i_WAdd"* address port is used to get the synaptic weight from the ROM. Second, the IF-Neuron integrates those weight and activity data. Afterwards, it compares the integration result to a predefined threshold to check the firing condition. In case the result is higher than the threshold, a spike is emitted by setting the *"Spike"* signal to high and resetting the output potential (*"Potent"*). Otherwise, *"Spike"* is low and the integration result is written to the RAM without modification. Next to that, the neuron's output *"Spike"* is used by the FIFO, as a write-enable, to save the firing logical neuron address given by *"Event"* signal. The spikes are then ready for reading by the next layer's NPM. Indeed, to read them through *"o_Event"* port, the next layer's NPM verifies the state of the FIFO by checking *"o_Empty"*. When an event is read, the *"i_Rd_Event"* is set high, which induces an update in the content of the FIFO buffer acted by a removal of the first event in its queue.

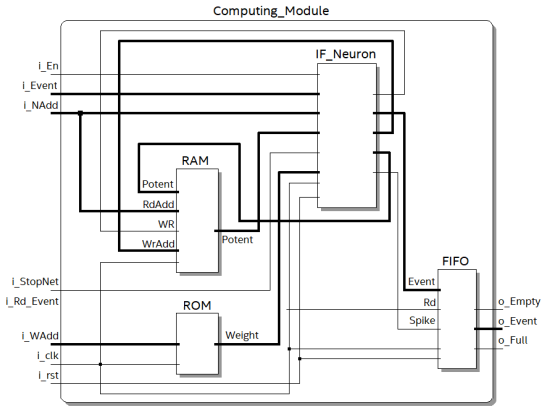

Fig. 8: *Computing_Core*'s internal structure.

3) Dense layers adaptation: With the dense layers, each neuron is connected to all its previous layer neurons. The dense NPM has, similarly to convolution NPMs, a control module and a computing module with similar functions. However, due to the connectivity difference, a change occurs at the level of the *Control_Module*, especially in the way it determines the neuron and weight addresses. The dense control module is also composed of an FSM and an address manager. However, the dense address manager is different, where the *Address_Compute* module is not used because the start/end indices, required for the address generation, are implicit with dense layers. This change implies a modification in the FSM, where the *"AddressCompute"* state is no longer necessary and, thus, bypassed. Therefore, a single zero to N-1 counter (N the number of logical neurons) is sufficient to get the neuron and weight addresses by the *Address_Generator*.

D. Terminate Delta Module

During SNN inference, the most spiking neuron of the last layer is selected as the winner class. In this work, we use the *terminate delta procedure* for enacting classification [5]. Figure 9 shows the *Terminate Delta Module* (TDM). It is composed of two maximum sub-modules that are used to determine the two most spiking neurons of the SNN last layer. The first and the second maximum blocks, *"Max1"* and *"Max2"*, give the number of spikes emitted by the first and second most spiking neurons. Next, a difference between these amounts is calculated to compare it to a delta value. Once the difference exceeds the delta value, the most spiking neuron is considered as the recognized class and provides a signal to halt the processing.

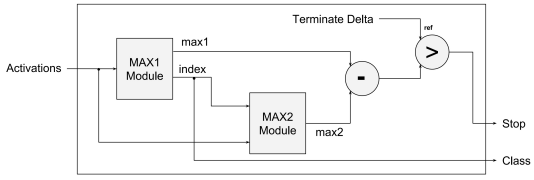

Fig. 9: Terminate Delta Module structure.

IV. Deployment on in-orbit satellite

A. CIAR and OPS-SAT

Since 2018, the CIAR project (Chaîne Image Autonome et Réactive / Responsive and Autonomous Imaging Chain) has been collaborating with the European Space Agency (ESA) on the OPS-SAT mission. OPS-SAT is a $3U$ ($30 \times 10 \times 10$ cm) nano-satellite that was launched on December 18, 2019. This is a demonstrator that aims to validate novel in-orbit control systems [14]. A camera with a sampling distance of 50 meters is embedded in OPS-SAT. A selected use case concerns cloud detection in images, which is a useful service compatible with this sampling distance. By identifying cloudy pixels, it can be decided to remove them at the source to preserve the satellite's memory and bandwidth. OPS-SAT embeds an Intel Cyclone-V System on Chip. We chose to use its programmable logic as the main computing resource in order to study neuromorphic hardware systems. The CIAR team designed algorithmic solutions with a high accuracy level, which are compatible with the number of Logic Elements (LE) available on this nano-satellite (110 kLE (Logical Element)). Several types of ANN hardware architectures are developed and deployed on OPS-SAT. On March 22, 2021, the team uploaded an ANN on the FPGA embedded in this in-orbit satellite [14]. A schematic representation of the experiment is illustrated in figure 10, where we can see that clouds have been segmented (colored in white on the left output segmentation map image) with a high level of accuracy. This demonstrates the advantage of an on-the-edge segmentation; it allows filtering images and transmitting only the cloud-free data to the ground-station. In this current work, we focus on SNN's deployment. In the following, we describe their deployment methodology and then analyze their in-flight execution results.

B. SNN's deployment methodology

The deployment methodology is composed of two steps: an *on-ground validation* phase and *in-orbit execution* phase.

First, in the *on-ground validation* step, the neural algorithms and their hardware implementations are scrutinized to validate

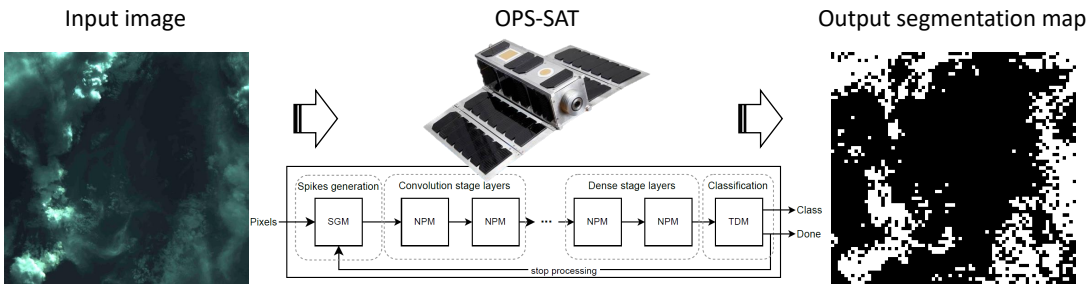

Fig. 10: Schematic representation of in-orbit experiment. The original input image is shown in the left and the result of its processing by SPLEAT is shown in the right of the figure.

them for the second in-orbit deployment phase. In this first preliminary validation phase, first, the ANNs are trained using *Tensorflow/Keras* libraries to generate a model descriptive ".h5" file. Then, using a Python script, this *".h5"* file is converted into a *VHDL package* (*".vhd"* file containing the ANN's topology, weights, ...). Next, this package is used to configure the ANN hardware architecture, which is described by a VHDL code. Afterwars, a functional validation using ModelSim tool is performed on the HDL architecture. This simulation step consists in verifying the behavior of the whole architecture on synthetic and real data. This is done by analyzing both the outputs FMs of the different layers and the final classification result. The metrics considered in this step are only the recognition rate and the latency. Once this step is successfully completed, the architectures are then ready for deployment on FPGA. In the last step, the ANN's architectures are integrated in a complete RTL design defined using the Qsys tool. The used Qsys interfacing platform is detailed in [24]. Next, using the Quartus tool, the hardware architectures are synthesized and bitstreams are generated. The bitstream is then loaded into a Cyclone's V FPGA, which is similar to the one, embedded in the orbital OPS-SAT satellite to perform on-ground cloud classification. Finally, the inference results are analyzed in the perspective of an in-orbit deployment.

The second *in-orbit execution* phase comprises four steps. First, a *Satellite Experimental Processing Platform* (SEPP) bitstream and a *Hard Processor System* (HPS) executable (".jar" file) are generated. Second, these files are tested and validated on an ESA's flat-Sat using both algorithmic and hardware metrics. For more description on the used metrics, refer to [14]. Next, the experiment is uploaded on OPS-SAT to perform an in-orbit execution. Finally, the data downloaded from the OPS-SAT satellite are analyzed and evaluated using the same metrics. The obtained results for this in-orbit deployment are presented in section V.

V. Experiment Results and Discussions

A. OPS-SAT dataset

In this paper, we essentially focus on the OPS-SAT cloud detection. Notice that, The used ANN models have been already tested and validated on the GTSRB and the MNIST datasets, refer to [4].

The OPS-SAT cloud segmentation dataset is composed of images that have been annotated by the CIAR project using a semi-manual pixel-level labeling technique. This dataset comes from the OPS-SAT satellite, which is located in a twilight sun-synchronous orbit. Therefore, the resulting images have a significant solar illumination variability, which affects the ANN performance, refer to [14] results. To perform segmentation using classification ANNs on this dataset, we use the same approach described in [14]. It consists in dividing the OPS-SAT data images into smaller patches and then attributing a unique label for each patch. The used ANN models are LeNet-like, thus, the images are divided into non-overlapping $28 \times 28 \times 3$ patches. Those are classified into two classes according to their cloud coverage. A cloud or non-cloud label is attributed by referring to their cloud coverage which ranges from 0% (cloud-free patch) to 100% (all-cloud patch). In this work and as in [14], we consider patches with less than 10% cloud coverage as non-cloud ones. Inversely, when this "10% cloud coverage" is exceeded, the patches are cloudy.

By doing so, the generated dataset, composed originally of 19 images with $2048 \times 1944 \times 3$ pixels, will gather 95703 of "$28 \times 28 \times 3$" classification patches. Then, 80% of those patches are used in the learning phase and the remaining 20% are used in the testing phase. Some of these dataset images and ground-truths are shown in figure 11. For more details about the data collection and their nature, refer to [14].

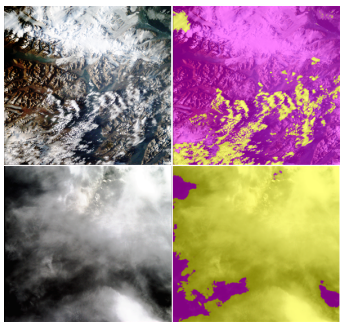

Fig. 11: OPS-SAT dataset examples. Top: dataset sample with cloud and snow. Bottom: dataset sample showing saturated and dark clouds. Left: RGB images. Right: images overlaid by a cloud ground-truth (yellow=cloud, magenta=no-cloud)

B. ANN models topologies

The ANN topologies used in this work, are illustrated in table I. They are LeNet-like CNNs with a $28 \times 28 \times 3$ input and with two output neurons representing the cloud and non-cloud classes. The first algorithm is a formal CNN with 1440 parameters. The second one is a Hybrid Neural Network (HNN) with a similar topology. This HNN aims to exploit both formal and spiking paradigms, its convolutional layers are implemented in the formal domain and the last layers are implemented in the spiking domain. No bias neurons are in the spiking dense layers, thus, it has less learning parameters (1428) than the formal CNN. Notice that, a Hard-Sigmoid activation function is used with the second convolution layer to make the output activities ranging between 0 and 1. Then after down-sampling step done by the last pooling layer, the output activities are converted by the SGM (see section III-B) to spike trains that are compatible with the last spiking dense layers. The last network is a LeNet-like spiking neural network (SNN) that has all its layers in the spiking domain. In this case of SNN, which is deployed later with SPLEAT, the convolution layers have strides that are equal to 2 instead of having pooling layers. Thus, the topology of the SNN is slightly different but with a similar structure: two convolution layers and two dense layers. This SNN has 2666 parameters.

CNN						
Layer	Units	Filters	Kernel	Stride	Bias	Activation
Conv2D	-	3	5x5	1x1	true	ReLU
Pool2D	-	-	2x2	2x2	-	-
Conv2D	-	5	5x5	1x1	true	ReLU
Pool2D	-	-	2x2	2x2	-	-
Dense	10	-	-	-	true	ReLU
Dense	2	-	-	-	true	SoftMax

HNN						
Layer	Units	Filters	Kernel	Stride	Bias	Activation
Conv2D	-	3	5x5	1x1	true	ReLU
Pool2D	-	-	2x2	2x2	-	-
Conv2D	-	5	5x5	1x1	true	H.Sigmoid
Pool2D	-	-	2x2	2x2	-	-
Dense	10	-	-	-	false	ReLU
Dense	2	-	-	-	false	SoftMax

SNN						
Layer	Units	Filters	Kernel	Stride	Bias	Activation
Conv2D	-	6	3x3	2x2	false	ReLU
Conv2D	-	6	3x3	2x2	false	ReLU
Dense	10	-	-	-	false	ReLU
Dense	2	-	-	-	false	SoftMax

TABLE I: ANN topologies

C. Model classification performances

In this section, we present inference results of the ANNs shown in table I when performing cloud segmentation based on patches classification on the described dataset. Notice that the output of classification ANNs (1×1) are upscaled to $28 \times 28 \times 1$ size output segmentation maps. Thus, processing all the patches of the original $2048 \times 1944 \times 3$ size image results in an output segmentation map with the same size and which can be compared with the ground-truth to get segmentation performance results. An output segmentation map example of an original real scale image is shown in figure 10.

Using the *"Tensorflow/keras"* library, we have built, learned and validated the ANN models on the training dataset. Then, we perform an inference phase on GPU platforms using a 32 bits floating-point precision. Afterwards, the models are converted to synthesizable VHDL codes to perform FPGA inference. The FPGA executions are then realized in a configurable fixed-point arithmetic, 16 bits ($FxP8.8$) with the HNN and the SNN^{1,2} and 17 bits ($FxP8.9$) with the CNN. The results of this FPGA inference in terms of F-score, precision and recall metrics are summarized in table II.

It is important to mention that the main objective of this work is to quantify the hardware deployment costs of the neural models, and not to enhance their performances. Hence, the objective here is to get results equivalent to those previously obtained on this OPS-SAT dataset in [14].

TABLE II: ANNs inference performances on FPGA.

Performance Metric	ANN model			
	CNN	HNN	SNN^1	SNN^2
Accuracy	56.83	69.83	**71.92**	70.78
F-score	46.90	62.72	**71.14**	68.86
Precision	61.38	**78.70**	73.53	74.12
Recall	37.95	52.14	**68.90**	64.30

We observe that the inference results of the spiking and hybrid neural networks, SNN^{1,2} and HNN, are more accurate than the formal CNN model when deployed on the FPGA. Indeed, in terms of accuracy these HNN and SNN^{1,2} models obtain scores of around 70%, whereas, the CNN obtains only 56.83%. A similar difference trend is observed when considering the other performance metrics. This shows the generalization ability and the robustness to the fixed-point arithmetic quantization of spike encoding. Moreover, these performances are quite similar to another *real-segmentation* convolutional network with a 4×4 output (FCN in [14]) which obtained on the same dataset the following scores: F-score=72%, precision=74% and recall=69%. Therefore, the models presented in this paper can be deployed on hardware due to their satisfactory results.

D. Hardware results

In this subsection, we present the hardware inference results of the ANNs presented in table I with a focus on resources usage, power consumption and execution time metrics.

1) Resource occupation: An important feature when dealing with the hardware implementation concerns the resources usage. The table III shows the post-routing results of the different ANN accelerators. In terms of memory blocks, the amount used by SPLEAT, implementing the SNNs, remains almost unchanged from the first version to the second one. This is because the memory blocks are used to save the synaptic weights and the neurons internal potential, which only depend on the ANN topology. In terms of registers, the second version of SPLEAT (SNN^2) has an increase in its occupation because of its parallel SGM that uses more memory inferred as registers. Finally, when considering ALMs usage, we notice that the two SNN versions occupy less amounts than the HNN

and the CNN. This ALM saving shows the benefit of spike coding which employs simpler computations. In addition, we notice an increase in the ALMs number used by SPLEAT from its first to second versions caused by the SGM parallelism. Indeed, the SGM with the second version processes the input pixel channels in a parallel way and therefore it employs more operators than the multiplexed version.

When considering the whole projects that contain the ANN accelerator entities, we notice similar resources usage tendency as with the isolated ANN accelerators one. Indeed, the integration environment of the ANN accelerators is constant from one project to another and, thus, occupies approximately a similar amount with all the different ANN models.

TABLE III: ANN accelerators post-routing resources usage (No DSP blocks are used).

Top level	Occupation (%)	CNN	HNN	SNN[1]	SNN[2]
Isolated Entity	ALMs	34.68	40.31	**5.24**	19.02
	Registers	25.27	32.18	**4.37**	8.11
	BRAM	**0**	0.06	1.94	1.88
	M10Ks	**0**	1.27	4.88	4.70
Total project	ALMs	59.96	62.49	**27.70**	40.43
	Registers	45.15	51.41	**23.73**	27.35
	BRAM	**4.67**	4.72	6.61	6.55
	M10Ks	**9.04**	10.31	13.92	13.74

2) Power consumption and latency performances: The in-orbit measured power consumption results of the different ANN accelerators and their corresponding estimations obtained using the Quartus tool are shown in table IV. The power consumption estimates of the HPS part are constant for all the synthesized networks, 1.39W and they are different from the measured ones. The reason for that is the fact Quartus does not use real scenarios but only makes simple hypotheses on the signals toggle rates of the circuits to simulate the functioning of the HPS part. Whereas during the on-board execution, a first preprocessing step is performed on the HPS part to prepare and communicate the images for the ANNs deployed on the Programming Logic (PL) of the FPGA. This explains the estimation versus measurement difference of about 0.8W.

When considering the PL part power consumption, we also observe a difference between the measurements and the estimations. Nevertheless, in this case, the differences are narrow and the power consumptions vary from one accelerator to another. Therefore, we consider that the estimation results are more representative on the PL part than on the HPS part of the FPGA. Indeed, we observe that the power consumption of the FPGA's PL part is related to the logic resources occupation (table III). The same trend occurs with the instantaneous power estimations and measurements, where the SNN consumes less power than the HNN and the CNN. Indeed, the SNN is the network that uses the least amount of logical resources. In fact, it occupies relatively fewer resources because it performs less instantaneous operations than the other architectures. Moreover, in the same logic, we observe that the network that consumes the highest instantaneous power is the HNN, which occupies the highest amount of logic resources on the FPGA. Notice that the SNN[2] has not been executed in-flight at that time. This constitutes a work in progress that depends on the satellite's availability for experiments.

TABLE IV: In-flight measured and Quartus estimated average power consumption of the ANN accelerators.

Entity	Metric	CNN	HNN	SNN[1]
HPS PDU (W)	Measured	2.53 ± 0.11	2.44 ± 0.09	2.40 ± 0.9
	Estimated	1.39	1.39	1.39
PL PDU (W)	Measured	1.72 ± 0.12	1.73 ± 0.13	1.64 ± 0.11
	Estimated	1.51	1.56	1.27
Total (W)	Measured	4.25 ± 0.24	4.17 ± 0.13	4.03 ± 0.16
	Estimated	2.90	2.96	**2.66**

The entity-isolated power consumption estimates and the measured inference processing times of the ANN whole projects are presented in table V. In terms of power consumption, the SNN deployed with SPLEAT having the time-multiplexed SGM (SNN[1]) consumes $5.91\times$ less than the CNN and $7.17\times$ less than the HNN. Moreover, as we have seen before, the SNN has also reduced the FPGA's logic footprint considerably. However, the execution time of the SNN ($1860\mu s$) is much longer than those of the other architectures as a result of its massively multiplexed architecture.

TABLE V: Isolated ANN accelerators' power consumption estimation and their measured inference processing times.

Hardware estimation	CNN	HNN	SNN[1]	SNN[2]
Dynamic Power (mW)	150.50	183.80	**28.11**	84.69
Total Power (mW)	243.99	295.99	**41.28**	201.50
Execution time (μs)	**25**	280	1860	160

To mitigate the latency, we have introduced some improvements to SPLEAT at the level of the spike generation module, which is explained in section III-B. In the first version, the spike generation module is implemented in a highly multiplexed fashion. Indeed, to verify the presence of spikes in a "real time step", the SGM goes through all the pixels with their 3 channels (RGB) of the patch. Therefore, it requires $28 \times 28 \times 3$ FPGA cycles to generate the spikes of one real time step per patch, which drastically increases the processing time of the architecture.

In the second version, we adopt a channel-level parallelism in which pixels R, G and B are processed in parallel. Theoretically, this allows dividing the processing time by 3, which in turn reduces the overall processing time. This has allowed it to reduce the timeout of the architecture, which allows it to stop the processing in case of non-convergence at the TDM level. In fact, these two modifications permit a reduction of the execution time by a factor of 11.6, passing from $1860\mu s$ with the multiplexed version to $160\mu s$ with the parallel one. Nevertheless, the power consumption increases when compared to the SNN[1]. On the other hand, the power consumption of this SNN[2] remains lower than those of the HNN and the CNN. Therefore, given its multiplexed structure and its lower power consumption than the formal networks, the

second version of SPLEAT represents an effective latency/area trade-off for embedded AI applications.

VI. Conclusion

The results obtained in this work highlight the advantages and limitations of adopting spiking neural networks in the context of orbital satellite image processing. In terms of hardware resources occupation, the SPLEAT architecture, in its massively multiplexed version, occupies almost $7\times$ less ALMs than the formal CNN architecture. Furthermore, SPLEAT with a parallel SGM occupies $1.8\times$ less than the formal CNN. However, when considering latency, a drastic increase in execution time is observed when using SPLEAT due to its multiplexed structure. Nevertheless, this architecture allowed a reduction in resources usage by a factor of about 6.62 compared to the formal CNN.

In order to mitigate the latency and to compromise it with the hardware resource usage, a second version of SPLEAT with a parallel SGM module was developed. This version reduced the execution time by a factor of $11.6\times$ with a marginal increase for resources (a factor of $3.69\times$). This parallelism allows the input pixel channels to be processed in parallel. In terms of algorithmic performance, spike coding showed a higher generalization capability than other architectures by achieving better FPGA inference results.

In summary, in this work we have demonstrated the feasibility of operating spiking neural networks on an FPGA on-board of an in-orbit satellite. The obtained results show a gain in terms of instantaneous power consumption and logic usage when using SNNs instead of formal CNNs. This enables improving the algorithmic performance of the SNN by using more FPGA resources to make the network deeper. In fact, we are currently working on this topic. Moreover, the latency limitations can be overcome by using spike-based learning algorithms, which are known to reduce the activity of the SNN. Furthermore, another improvement of SPLEAT would be the use of event-driven sensors that generate event-based data that are well suited to SNNs. Thus, coupling such asynchronous sensors to SPLEAT represents a promising approach for embedded AI applications.

Acknowledgement

This work is funded by the French institutions *IRT Saint Exupéry*, *Université Côte d'Azur* and *Région Sud Provence-Alpes-Côte d'Azur*. We would like to thank the *CIAR project* members and partners (Thales Alenia Space, Activeon, Avisto, Elsys Design, MyDataModels, TwinsWheel, Geo4i, LEAT and INRIA) and ESA for their contributions in getting the results of this work.

References

[1] C. Lee, S. S. Sarwar, P. Panda, G. Srinivasan, and K. Roy, "Enabling spike-based backpropagation for training deep neural network architectures," *Frontiers in Neuroscience*, vol. 14, p. 119, 2020.

[2] L. Abbott, "Lapicque's introduction of the integrate-and-fire model neuron (1907)," *Brain Research Bulletin*, vol. 50, pp. 303–304, 1999.

[3] J. A. Perez-Carrasco *et al.*, "Mapping from Frame-Driven to Frame-Free Event-Driven Vision Systems by Low-Rate Rate Coding and Coincidence Processing–Application to Feedforward ConvNets," *IEEE PAMI*, vol. 35, no. 11, pp. 2706–2719, 2013.

[4] N. Abderrahmane and B. Miramond, "Neural coding: adapting spike generation for embedded hardware classification," in *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2020.

[5] N. Abderrahmane, E. Lemaire, and B. Miramond, "Design space exploration of hardware spiking neurons for embedded artificial intelligence," *Neural Networks*, vol. 121, pp. 366 – 386, 2020.

[6] D. F. Goodman and R. Brette, "Brian: a simulator for spiking neural networks in python," *Frontiers in neuroinformatics*, vol. 2, p. 5, 2008.

[7] P. Qu, Y. Zhang, X. Fei, and W. Zheng, "High performance simulation of spiking neural network on gpgpus," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 11, pp. 2510–2523, 2020.

[8] A. Das *et al.*, "Mapping of local and global synapses on spiking neuromorphic hardware," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1217–1222, IEEE, 2018.

[9] A. Joubert, B. Belhadj, O. Temam, and R. Héliot, "Hardware spiking neurons design: Analog or digital?," in *International Joint Conference on Neural Networks*, 2012.

[10] F. Akopyan *et al.*, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, 2015.

[11] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[12] L. A. Camunas-Mesa *et al.*, "A Configurable Event-Driven Convolutional Node with Rate Saturation Mechanism for Modular ConvNet Systems Implementation," *Frontiers in Neuroscience*, vol. 12, 2018.

[13] S. Moradi, Q. Ning, F. Stefanini, and G. Indiveri, "A scalable multi-core architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps)," *CoRR*, vol. abs/1708.04198, 2017.

[14] F. Férésin, E. Kervennic, Y. Bobichon, E. Lemaire, N. Abderrahmane, G. B. I. Grenet, M. Moretti, and M. Benguigui, "In space image processing using ai embedded on system on module: example of opssat cloud segmentation," *2nd European Workshop on On-Board Data Processing*, 2021.

[15] N. Abderrahmane and B. Miramond, "Information coding and hardware architecture of spiking neural networks," in *Euromicro Conference on Digital System Design (DSD)*, pp. "1–8", 2019.

[16] H. Mostafa, "Supervised learning based on temporal coding in spiking neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 7, pp. 3227–3235, 2018.

[17] Y. Cao, Y. Chen, and D. Khosla, "Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition," *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54–66, 2015.

[18] A. S. Cassidy *et al.*, "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores," in *International Joint Conference on Neural Networks*, 2013.

[19] E. Izhikevich, "Which Model to Use for Cortical Spiking Neurons?," *IEEE Transactions on Neural Networks*, vol. 15, pp. 1063–1070, 2004.

[20] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, "STDP-based spiking deep convolutional neural networks for object recognition," *Neural Networks*, vol. 99, pp. 56–67, 2018.

[21] L. Khacef, N. Abderrahmane, and B. Miramond, "Confronting machine-learning with neuroscience for neuromorphic architectures design," in *International Joint Conference on Neural Networks (IJCNN)*, 2018.

[22] S. Thorpe and J. Gautrais, "Rank order coding," in *Computational neuroscience*, pp. 113–118, Springer, 1998.

[23] N. Abderrahmane, *Hardware design of spiking neural networks for energy efficient brain-inspired computing*. Phd thesis, Université Côte d'Azur, Dec. 2020.

[24] F. Feresin, M. Benguigui, Y. Bobichon, E. Lemaire, M. Moretti, and G. Bahl, "On board images processing using ia to reduce data transmission: example of opssat cloud detection," in *7th On-Board Payload Data Compression Workshop, OBPDC*, 2020.