Modélisation et exploration d'architectures neuromorphiques pour les systèmes embarqués haute-performance

Edgar Lemaire

▶ **To cite this version:**

Edgar Lemaire. Modélisation et exploration d'architectures neuromorphiques pour les systèmes embarqués haute-performance. Réseau de neurones [cs.NE]. Université Côte d'Azur, 2022. Français. NNT : 2022COAZ4009 . tel-03665844v1

HAL Id: tel-03665844

https://theses.hal.science/tel-03665844v1

Submitted on 12 May 2022 (v1), last revised 24 Apr 2023 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THALES

THÈSE DE DOCTORAT

Modélisation et Conception d'Architectures Neuromorphiques pour les Systèmes Embarqués Haute-Performance

Edgar LEMAIRE

Laboratoire d'Antennes, Électronique et Télécommunications (LEAT) &
Thales Research & Technology (TRT-Fr)

Présentée en vue de l'obtention du grade de docteur en Electronique **d'**Université Côte d'Azur

Dirigée par : Benoît Miramond, LEAT
Co-Dirigée par : Sébastien Bilavarn, LEAT
Co-Encadrée par : Hadi Saoud, Thales
Soutenue le : 08 Mars 2022

Devant le jury, composé de :
Simon Thorpe, Dir. de Recherche, CerCO
Gilles Sassatelli, Dir. de Recherche, LIRMM
Olivier Bichler, Ingé. Chercheur, CEA-LIST
Jean Martinet, Professeur, I3S
Philippe Millet, Resp. pôle Innovation, Nexter
Hadi Saoud, Ingé. Chercheur, Thales
Benoît Miramond, Professeur, LEAT
Sébastien Bilavarn, Maitre de Conf., LEAT

Cifre

ANRT
ASSOCIATION NATIONALE RECHERCHE TECHNOLOGIE

Modélisation et Conception d'Architectures Neuromorphiques pour les Systèmes Embarqués Haute-Performance

Jury :

Rapporteurs

Simon Thorpe, Directeur de Recherche, Centre de Recherche Cerveau et Cognition (CerCO) - CNRS/Université Toulouse 3 Paul Sabatier

Gilles Sassatelli, Directeur de Recherche, Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM) – CNRS/Université Montpellier 2

Examinateurs

Jean Martinet, Professeur des Universités, Laboratoire d'Informatique, Signaux et Systèmes de Sophia antipolis (I3S) – CNRS/Université Côte d'Azur

Olivier Bichler, Ingénieur Chercheur, Laboratoire d'Intégration des Systèmes et Technologies (LIST) – CEA Tech

Philippe Millet, Directeur de l'Innovation, Nexter

Hadi Saoud, Ingénieur Chercheur, Thales Research & Technology (TRT), Thales

Benoît Miramond, Professeur des Universités, Laboratoire d'Eléctronique, Antenne et Télécommunications (LEAT) – CNRS/Université Côte d'Azur

Sébastien Bilavarn, Maître de Conférences, Laboratoire d'Eléctronique, Antenne et Télécommunications (LEAT) – CNRS/Université Côte d'Azur

Résumé en Français

Les réseaux de neurones profonds ont permis des progrès sans précédent dans le domaine de l'apprentissage automatique. En imitant le calcul parallèle et distribué du cerveau, de tels algorithmes permettant en effet d'émuler n'importe quelle fonction, jusqu'aux plus complexes. Leurs applications vont de la vision par ordinateur (classification, segmentation, détection...), au traitement du language naturel en passant par la prédiction de séries temporelles. Dans tous ces domaines, les réseaux de neurones profonds n'ont cessé de repousser les limites de l'intelligence artificielle. D'autre part, nous assistons par ailleurs à l'émergence progressive de l'internet des objets (Internet of Things, IoT). Ces systèmes embarqués connectés ont un besoin grandissant en capacité de traitement, et les réseaux de neurones profonds semblent alors tout indiqués pour accomplir cette tâche. Qu'il s'agisse de navigation autonome dans des drones ou dans des voitures sans pilote, de reconnaissance faciale dans les téléphones portables ou plus généralement de traitement périphérique (Edge Computing) dans les réseaux de capteurs, les applications embarquées de l'apprentissage profond sont déjà nombreuses. Cependant, les algorithmes neuronaux sont complexes et particulièrement gourmands en ressources de calcul. En matière de consommation énergétique, ces derniers semblent en effet incompatibles avec la nature contrainte des systèmes d'IoT. Dans cette optique, l'approche de l'Électronique Neuromorphique consiste à s'inspirer du cerveau biologique pour en mimer l'efficacité énergétique. En effet, il s'agit du modèle de processeur neuronal le plus abouti à notre connaissance, affichant une consommation d'à peine 20 watts. À travers l'utilisation de modèles de neurones bio-inspirés proposés par les neurosciences computationnelles, l'approche Neuromorphique a pour objectif de réduire la consommation énergétique de l'intelligence artificielle embarquée. Cette approche repose sur des arguments prometteurs: la simplicité du calcul impulsionnel, l'aspect binaire des synapses et l'encodage "épars" (sparse). Cependant, la littérature scientifique spécialisée semble manquer de comparaison étendue de l'impact du domaine de codage sur l'efficacité énergétique des accélérateurs neuronaux. De ce fait, la contribution principale de cette thèse est une comparaison détaillée des domaines de codage impulsionnels et formels sur FPGA. Cette étude implique plusieurs cas d'usage et différents niveaux de parallèlisme. Ce volet de la thèse a été mené a bien au moyen du développement d'un estimateur de ressources et d'énergie pour les architectures neuronales sur FPGA. Cet outil permet d'accélérer l'exploration de l'espace des applications. D'autre part, nous proposons de quantifier les gains potentiels au moyen d'une métrique et d'un modèle de haut niveau: le ratio d'activité synaptique. Cette métrique permet d'évaluer rapidement les gains potentiels d'énergies offerts par le domaine impulsionnel pour une application donnée. Nous l'utilisons notamment pour trouver de nouvelles familles de réseaux impulsionnels prometteuses en termes énergétiques. Enfin, nous mettons à profit les enseignements de cette étude pour proposer l'accélération neuronale hybride, une architecture mélant les deux domaines d'encodage (impulsionnel et formel). L'architecture a été embarquée à bord du satellite OPS-SAT (lancé en Décembre 2019) et testée en vol. Ce faisant, il s'agit de la toute première architecture neuromorphique fonctionnelle dans l'éspace.

Mots Clefs - Réseaux de Neurones Artificiels, Réseaux de Neurones Impulsionnels, Eléctronique Numérique, Architecture Matérielle, Modélisation, Consommation énergétique, Systèmes Embarqués, Informatique en Périphérie, Eléctronique Neuromorphique, FPGA

Abstract

Deep Neural Networks have recently pushed unprecedented progress in the field of Machine Learning. Those algorithms mimic the parallel and distributed computation paradigm of the biological brain to fit any desired functions, including the most complex ones. The range of applications for Deep Learning is very wide: from computer vision (classification, segmentation, detection...) to natural language processing and time-series prediction. Those domains were all greatly impacted by Deep Neural Networks, since they are continuously pushing back the limits of Artificial Intelligence. On the other hand, the emergence of the Internet of Things in recent years have opened a brand new range of applications for Deep Learning. Indeed, the processing abilities of those algorithms is particularly appealing for such autonomous systems like IoT. From navigation and obstacle detection in drones and self-driving cars, to face recognition in smartphones, or more generally for Edge Computing in networks of sensors, Deep Learning seems all indicated for the task. However, Deep Learning models are very complex and energy-intensive. In terms of energy consumption, they appear not to comply with the highly-constrained nature of embedded systems. To cope with this issue, the Neuromorphic Computing approach consists in taking inspiration from the biological brain. Our brain is indeed the most efficient neural processor anyone has ever heard of, with only 20W of average power consumption. By using bio-inspired models from Computational Neurosciences, the Neuromorphic approach thus aims in reproducing the energy efficiency of the brain. This choice is backed by several objective assessments, like the hardware-friendliness of the spiking synaptic operation, the sparse computation or the lightweight communication between neurons. However, the literature seems to lack broad and fair comparisons of the impact on neural coding on the energy consumption of hardware neural networks. Therefore, the first contribution of this work is a fair and extensive comparison between spiking and formal coding domains on FPGA. The study also takes low-level implementation into account, like parallelism. It was made possible by the development of an hardware-footprint estimator for hardware neural networks on FPGA, which drastically facilitates and accelerates the design and applications space exploration. Moreover, we propose an high-level metric and energy model to evaluate the energy consumption of neuromorphic accelerators. This model named Synaptic Activity Ratio (SAR) is based on the number of spikes per synapse and the ratio of energy consumption between spiking and formal synaptic operations. The model is indeed able to determine whether a given CNN model (tested on a given dataset) could bring energy savings on FPGA. This contribution also facilitates the exploration of applications relatively to neural coding domains. For instance, we also use this technique to find novel SNN representations which seem better adapted to hardware acceleration than the basic rate-coded conversion approach. Finally, we use the knowledge acquired from this study to introduce the novel concept of neural coding hybridization. We propose an hardware architecture which uses both formal domain for feature-extraction and spiking domain for classification. The architecture was embedded on-board OPS-SAT satellite (launched Dec. 2019) for in-flight test, being the world first neuromorphic architecture in space.

Keywords - Artificial Neural Networks, Spiking Neural Networks, Digital Electronics, Hardware Architecture, Modeling, Energy Consumption, Embedded Systems, Edge Computing, Neuromorphic Electronics, FPGA

Acknowledgements

I would like to express my deepest gratitude to my supervisors: Benoît Miramond, Sébastien Bilavarn, Hadi Saoud and Philippe Millet for their unwavering help and support during the three years of my thesis. I would like to thank them for believing in me with this project although I had very little experience and knowledge on digital electronics; for teaching me so much professionally, technically and humanely; and for the support they brought me all along those three years.

I would also like to express my gratitude to Simon Thorpe, Gilles Sassatelli, Olivier Bichler and Jean Martinet for accepting to be part of my thesis jury. It is a great honor for me to defend my work in front of such prominent researchers.

I would also like to thank my colleagues of both LEAT and Thales RT for their support and the many friendship we built. In particular, I would like to sincerely thank Nassim Abderrahmane for all the work we have accomplished together, and all the help he offered me in the early stages of my thesis.

In addition, I would like to thank my parents Frédérique and Bruno Lemaire, my sister Félicie Lemaire, my grand-parents, uncles, aunts, cousins and friends for their support, encouragement and prodding. Each one of them had a role to play in the achievement of this work. I would also like to thank Manon Philip for her unwavering support and love, which helped me going forward during hard times.

This work was funded by Thales and ANRT (Agence Nationale de la Recherche Technologique), and hosted at both Thales Research Technology's LCHP (High Performance Computing Laboratory) and CNRS & Université Côte d'Azur LEAT (Electronics, Antennas and Telecommunication Laboratory).

List of Publications

Journal Papers

Abderrahmane, N., Lemaire, E., & Miramond, B. (2020). Design space exploration of hardware spiking neurons for embedded artificial intelligence. Neural Networks, 121, 366-386.

Lemaire, E., Miramond, B., Bilavarn, S., Saoud, H. & Abderrahmane, N., (2022). Synaptic Activity and Hardware Footprint of Spiking Neural Networks in Digital Neuromorphic Systems. ACM Transactions on Embedded Computing Systems (Accepted for publication)

Conference Papers

Lemaire, E., Moretti, M., Daniel, L., Miramond, B., Millet, P., Feresin, F., & Bilavarn, S. (2020, October). An FPGA-based Hybrid Neural Network accelerator for embedded satellite image classification. In 2020 IEEE International Symposium on Circuits and Systems (ISCAS) (pp. 1-5). IEEE.

Book Chapters

Chapter 12 : Space Use-Case, in the book :
Jahre, M., Göhringer, D., & Millet, P. (Eds.). (2021). Towards Ubiquitous Low-power Image Processing Platforms. Springer International Publishing.

Workshop Papers

Férésin, F., Kervennic, E., Bobichon, Y., Lemaire, E., Abderrahmane, N., Bahl, G., ... & Benguigui, M. (2021). In space image processing using AI embedded on system on module: example of OPS-SAT cloud segmentation. In 2nd European Workshop on On-Board Data Processing.

Patents

FISO neuron model, submitted in September 2019, in proceeding.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

In recent years, the field of artificial intelligence has been widely dominated by Deep Learning. This field of Machine Learning uses Artificial Neural Networks which model the biological neural networks in mathematical equations. This approach enables to emulate more and more complex cognitive functions in automated algorithms, such as image classification, object detection, natural language processing or decision making. Since it first appeared in the foreground of research in the 90's, all those domains have been revolutionized by Deep Learning. The complexity of the models growing further and further, the limits of Machine Learning are constantly pushed back, involving more and more application domains and industrial actors.

1.1 History of Deep Learning

Long before its modern supremacy, the premises of Deep Learning dates back to the very beginning of computer science. In the same school of thought of Alan Turing, researchers like McCulloch and Pitts worked to demonstrate that cognition could be modeled through mathematical functions. In 1943, they proposed the first artificial neuron model [22]. In this breakthrough publication, the authors demonstrated that a network of such formal neurons was able to emulate propositional logic functions. Later in 1957, Rosenblatt proposed the first application of formal neurons to automated classification: the perceptron [23]. The perceptron is made of a single layer of formal neurons. The major contribution of this pioneering publication is the learning algorithm. By iterating over a set of labeled training samples, the algorithm automatically adjusts the synaptic weights to minimize the output error. In doing so, the authors proposed the first supervised learning algorithm. However, the technique was limited to single layers of neurons, which were only able to solve binary classification problems. This limitation was leveraged in another major breakthrough when Werbos proposed the Multi-Layer Perceptron (MLP) and gradient BackPropagation (BP) algorithm in 1974 [24]. The MLP is considered by many as the first modern neural network model, and the BP algorithm is still a fundamental element of modern Machine Learning. The BP algorithm allows to train several successive perceptron layers at once, hence the name Multi-Layer Perceptron. The deeper meaning of this work is even more revolutionary: an MLP of arbitrary size and depth is theoretically able to solve any non-linear problem. However, the feasibility of this algorithm was limited by the available computing power in the early 70's.

Following Moore's law, the exponential growth of computing power made it possible in 1986, when Rumelhart proposed the first implementation of the BP algorithm in a feed-forward MLP [25]. This work opened the gates to what has since become modern Deep Learning by using deeper and deeper networks, the models were able to solve more and more complex tasks in various application fields. The most prominent of which was computer vision: extracting information from images for classification, segmentation or detection. In 1989, LeCun proposed the first Convolution Neural Network (CNN) [26] trained with the BP algorithm. The innovation of this model is to give a structure to the layers of neurons according to specificities of the application. For computer

vision, the convolution layers are designed to extract spatial patterns. LeCun later generalized the CNN model to various tasks in the fields of image classification, speech recognition and time-series prediction [27]. The convolution layers are still widely used in Deep Learning to this date, and are one of the key components of modern neural network architectures like VGG-16 [28]. In recent years, other Formal Neural Networks (FNN) models have emerged for various domains of application, like Transformer neural networks for Natural Language Processing [29]. In doing so, the models are getting deeper and deeper and more and more complex. For example, the recent GPT-3 transformer model proposed by the OpenAI team in 2020 [30] uses 175 billions of parameters, *i.e.* 175 billions of explicit synaptic connections.

1.2 Energy Consumption and Embedded Systems

Neural network models becoming more and more complex, they have higher and higher computational requirements for training and inference. For example, the GPT-3 transformer model with 175 billion parameters requires a bare minimum of 350GB of GPU memory for inference. That is equivalent to 8 Nvidia A6000 GPUs; the latest and most powerful deep-learning oriented High-Performance Computing (HPC) device. This represents a power consumption of 2400W. Therefore, the energetic scalability of Machine Learning starts to raise questions among the scientific and public communities. But this energetic issue is even more problematic in the field of embedded systems. Indeed, the recent emergence of the Internet of Things (IoT) opened a brand new field of applications for neural networks. From autonomous navigation in drones or self-driving cars, to advanced robotics and augmented reality, the need for such embedded artificial intelligence is just starting to rise. By definition, those systems are isolated from the power grid, and must run on battery or energy harvested from the environment (mostly solar panels). Quite straightforwardly, the energy budget of Deep Learning is not compliant with such constrained systems.

An approach chosen by many researchers in the field of neural network acceleration is to take inspiration from the most advanced and efficient neural processor anyone has ever heard of: the biological brain. The human brain is said to feature more than 10 000 billions synaptic connections per cm^3. On the other hand, it consumes no more than 20W in average. Since the volume of human brain is roughly $1450cm^3$, that is an average of $10^{-17}W$ per synapse. For comparison, the most advanced neural network model (GPT-3) running on the most advanced hardware available (Nvidia A6000) uses approximately $10^{-8}W$ per synapse (175 billions synapses and 2400W for 8 A6000 GPUs). That is, if we omit the nature of the network itself (its topology and actual neuron dynamics) and if we consider that the number of synapses is a reliable metric to compare the brain and a deep learning model, the first is approximately one billion times more energy-efficient than the latter. It seems that we indeed have a lot to learn from biology regarding low-power neural processing.

1.3 Bio-inspired neurons for Machine Learning

In the field of Machine Learning, the main interest is to develop powerful models applied to solving real-world problems, like image segmentation for self-driving cars or natural language processing for virtual assistants. However, another field of research has long paid interest to artificial neural networks for other motivations. Neurosciences and Cognitive Sciences use them as a tool to

simulate cognitive functions, providing a better understanding of the underlying mechanisms of the human mind. For this purpose, computational Neurosciences are interested in the biological plausibility of the neuron models. In doing so, many biological neuron models have been proposed, mimicking the event-based nature of the brain. In contrast with Machine Learning which uses static real-valued information encoding, the biological brain uses action potentials: brief electrical impulses better known by the name of spikes. In Spiking Neural Networks (SNNs), information is encoded temporally with spikes. The strong hypothesis at the basis of our approach is the following: spike encoding is one of the major sources of power efficiency in the biological brain, compared to formal Machine Learning models. This hypothesis is backed by several factual assessments. In the widely used Integrate & Fire spiking neuron model, multiplication-accumulation operations are replaced with accumulation. The latter uses fewer logic and power than the first [31], hence the expectations of a better hardware-efficiency. Moreover, the sparse encoding of information in SNNs enables a sparse computation. In other words, the design is only active upon processing input spikes and remains idle otherwise. This feature is known as "frame-free", in contrast with the frame-constrained nature of the formal approach. Additionally, spikes enable light-weight communication between neurons, further reducing the hardware footprint.

In the light of these assessments, a recent trend in the field of neural network acceleration is to use bio-inspired neurons in deep-learning network models. It consists in developing application specific hardware architectures to support the inference of deep SNN models. By using bio-inspired neuron and event-based computation, expectations are a major reduction of energy consumption. This is a particularly appealing promise for embedded systems. In all, SNNs are expected to leverage the energy consumption limitations of embedded artificial intelligence. The thesis lies precisely in this field of the research: Neuromorphic Computing. More precisely, this work deals with the hardware acceleration of Spiking Neural Networks on FPGA, and the associated energy savings.

1.4 Problem statement and outline

However, there is still a lack of clear, fair and extensive comparison between formal and spiking hardware neural networks in the literature. The hardware-efficiency promises remain unproven outside of very specific cases, and most importantly: to be quantified. In this thesis, we address the most widely used type of SNNs to our knowledge: rate-coded SNNs converted from formal models. The goal of this thesis is to evaluate the potential energy efficiency gains of this model over conventional Formal Neural Networks in the context of FPGA accelerators. More precisely, this thesis aims in providing a cartography of neural coding domains and applications. In other words, the goal is to determine which type of application would be better suited to spiking domain acceleration in terms of energy gains. Moreover, this study follows a hardware-software co-design approach: the model and the underlying hardware has to be designed according to one another. In doing so, our goal is also to find innovative ways to benefit from spiking neurons in hardware and *vice versa*. The contributions of this work relatively to the literature will be described further at the end of the State-of-the-Art section (Chapter 2).

This document begins with a detailed state-of-the-art (Chapter 2), including a theoretical background of formal and spiking neural network models and hardware acceleration. We also provide an overview of the existing architectures and few publications addressing the comparison of the two neural coding domains. In Chapter 3, we describe our prototype of hardware accelerators for fully-connected spiking layers: PADS. It is based on rate-coding, and is used as an experimental

testbench to study the energy consumption of neuromorphic systems. In Chapter 4, we propose the Synaptic Activity Ratio (SAR) metric and the associated high-level energy estimation model. The model is used to rapidly determine whether a model or application is suitable for spiking acceleration, in terms of energy savings over formal acceleration. In Chapter 5, we propose an estimation framework based on low-level measurements and FPGA modeling. The framework enables rapid estimation of resources, power usage, inference time and energy consumption of formal and spiking accelerator under high and low level of parallelism. The framework enables a fast and reliable cartography of application, neural coding domain and implementation choices. It is applied to a benchmark of datasets, and the energy estimations are confronted with the SAR-based model. In Chapters 6 and 7, we propose two hardware implementations based on PADS architecture. In the light of the previous studies, those architectures intends to leverage the potential energy savings expected from neuromorphic acceleration. In Chapter 6, we propose neural network hybridization in order to tailor the neural coding domain according to synaptic activity at layer level. In Chapter 7, we use our acquired knowledge and models to find a more suitable SNN representation for neuromorphic acceleration. That is, a novel representation in which SNNs are viewed as Recurrent Neural Networks on a finite (and defined) number of timestep [32]. Using SoD spike encoding [33] and Surrogate Gradient Learning, this new model of SNNs achieve lower SAR than rate-based SNNs for even accuracy. We also propose an adaptation of the PADS architecture to this new model.

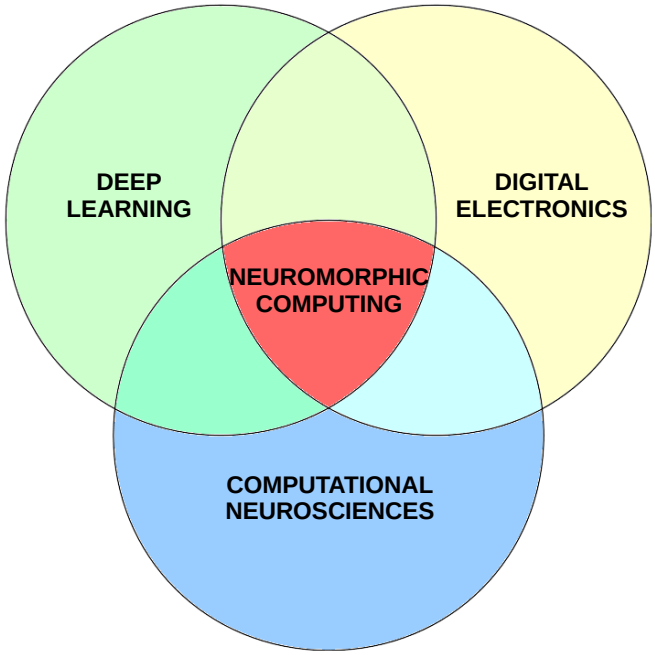
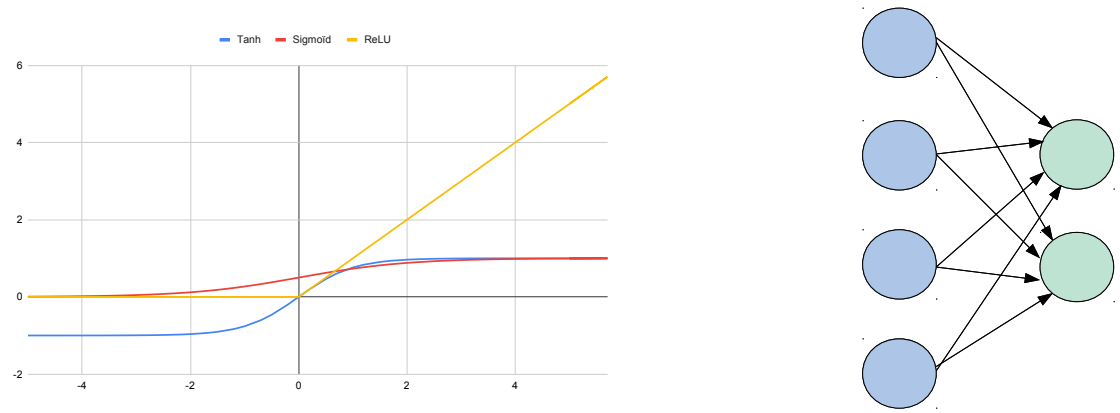


Figure 1.1: Neuromorphic Computing is a trans-disciplinary approach between Deep Learning, Digital Electronics and Computational Neurosciences.

Chapter 2

State of the Art & Contributions

Chapter contents



(a) (b)

Figure 2.1: Left: commonly used activation functions in formal neurons: hyperbolic tangeant (Tanh), sigmoïd and Rectified Linear Unit (ReLU). Right: Schema of FC layer, with 5 input and 2 output neurons.

Before going into more details on the works and results obtain to achieve the thesis goals, we briefly introduce the State of the Art of neuromorphic accelerators. First, we provide information on conventional Formal Neural Networks (FNN) and their hardware implementations. Then, we introduce Spiking Neural Networks (SNN) and how they differ from classical FNNs. More precisely, we are going to explain how those differences are expected to bring resource, power and energy savings to neuromorphic computing. In doing so, we describe some existing SNN accelerators and techniques aiming to benefit from the spiking domain promises.

2.1 Formal neurons

Formal Neurons are very roughly inspired from biological models in their behaviour. They are designed to integrate incoming weighted activations from several uphill neurons, just like the biological neuron does with incoming action potentials in its soma. Each of those activations are weighted by a synaptic-weight. From this "soma activity", which is also referred to as "membrane potential", an output activation is computed thanks to an activation function. This output activation is then propagated to downhill neurons. This behavior is summarized in Equation 2.1.

$$y_j^l = f(s_j^l), \quad s_j^l = \sum_{i=0}^{N_{l-1}-1} w_{ij} * y_i^{l-1} \tag{2.1}$$

With y_j^l being the output of the j^{th} neuron of layer l, s_j^l the membrane potential of the j^{th} neuron of layer l and w_{ij} the synaptic weight between the i^{th} neuron of layer $l-1$ and the j^{th} neuron of layer l, and $f()$ the non-linear activation function. The activation function is usually a non-linear function such as hyperbolic tangent (Tanh), sigmoïd or Rectified Linear Unit (ReLU). Those activations functions are shown in Figure 2.1a.

However, this single artificial neuron is not useful in itself. It starts to gain interest when using several interconnected neurons, through which information flows successively. This set of interconnected neurons is what is called an Artificial Neural Network.

2.1.1 Feed-forward NN and CNNs

Artificial Neural Networks come in various types and families. Those types are called topologies, which characterize the organization of neurons in the network. In the present work, we deal with feed-forward, layer-based neural networks. If some types of networks are unstructured, such as reservoir neural networks [34] or liquid state machines ([35]), feed-forward neural networks are formed of successive layers, through which information flows from the input to the output, always in the same direction. Therefore, we will not cover topologies such as Recurrent Neural Networks, in which activation does not only flows downstream, but can be branched to itself or upstream layers. A layer is a set of neurons, which are connected to neurons of the previous and following layers. In the Machine Learning community, the layer corresponds more to the connection scheme between two set of neurons, than the set of neuron in itself. This connection policy will indeed serve to emulate a transformation of information between two set of neurons. The most basic type of layer is the Fully-Connected (FC) layer. An illustration of FC layers is shown in Figure 2.1b.

It implies an all-to-all connection between upstream and downstream neurons of a layer. This function is used to classify data, and can be used on its own to build a Multi-Layer Perceptron (MLP), which is composed of several successive FC layers. It has been stated by Hornik et. al. [36] that an MLP composed of two fully-connected layers was sufficient to approximate any non-linear separable function with an arbitrary precision, *i.e.* it is sufficient to solve any classification task. However, when using only FC layers, complex classification tasks (*i.e.*, with a lot of classes, large images, complex objects, rotations...) requires an incredible amount of neurons in those layers. When adding layers, the problem might be solved with fewer neurons, but this still requires a lot of neurons and connections, which implies a huge computation and memory requirements. In order to solve this problem, more specific layers have been proposed by LeCun et. al. [37] in 1980: Convolutional and Pooling Layers. Convolutional layers emulate a convolution operation using one or several filters. This type of layer is well suited for the extraction of spatial or temporal patterns. Moreover, this layer implies a lower number of synaptic weights, as they are shared among filters. A convolution layer with 5 filters of size 3×3 only has $3 \times 3 \times 5 = 45$ weights, which helps mitigating memory requirements of Artificial Neural Networks. The Convolutional layer is represented in Figure 2.2. In this figure, two steps of the Convolutional process are represented. The first step is the convolution of the orange patch with the filter, which results in the orange activation in the output feature map. The last step is also shown in blue. All intermediate steps are omitted for clarity. Pooling layers on the other hand emulate a sub-sampling operation. This is used to mitigate the computation and memory requirements of Artificial Neural Networks, as Convolutional layers may result in large output size due to a large number of filters or large input vector size. Similarly to Convolutional layers, Pooling layers associate an output neuron to an input patch. However, the output activation is not obtained by convolving a filter, but rather using Maximum or Average operations: Max-Pooling layers compute the output activation as the maximum value of the associated input patch, whereas Average-Pooling layers compute the output activation as the average of the patch. The combination of Convolution and Pooling layers form a Feature-Extraction (FE), which is usefull to pre-process the data before classification through FC layers. Basically, a typical Convolutional Neural Network such as LeNet5 [37] (Figure 2.3) is composed of a FE stage followed by an MLP.

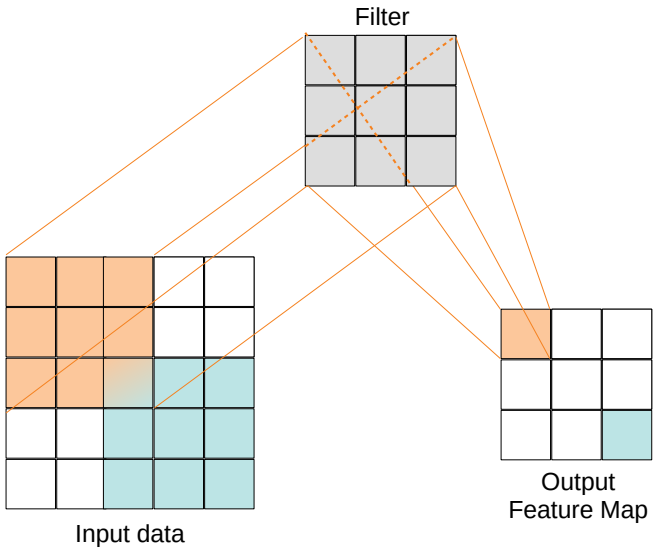


Figure 2.2: Schema of a Convolutional layer, with a 5x5x1 input data, a single 3x3 filter and a stride of 1.

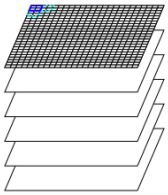
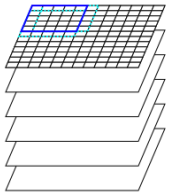
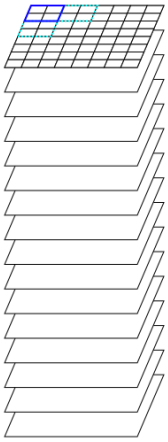
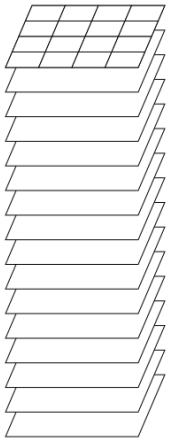
Layer	Input	Conv 1	Pool 1	Conv 2	Pool 2	D 1	D 2	D 3
Hyper Parameters		$N_{filters} = 6$ $W_{filters} = 5$ $S = 1$	$N_{filters} = 6$ $W_{filters} = 2$ $S = 2$	$N_{filters} = 16$ $W_{filters} = 5$ $S = 1$	$N_{filters} = 16$ $W_{filters} = 2$ $S = 2$	$N_{out} = 256$	$N_{out} = 84$	$N_{out} = 10$
Out size	28x28x1	24x24x6	12x12x6	8x8x16	4x4x16	256	84	10
Illustration								

Figure 2.3: Summary of LeNet-5 CNN. Conv is for Convolutional Layer, Pool is for Pooling layers and D is for FC layers. The visuals are obtained using N2D2 sotfware [1]

2.1.1.1 Advanced layers

In recent years, new types of layers emerged in addition to the three aforementioned fundamental layer structures. A short list of the most important advanced layers is given below for information.

- Batch-Normalization layer [38]: normalizes activation to stabilize and speed-up learning
- Recurrent layers [39]: short term memory for temporal signal processing
- Residual layer [40]: recurrent connection across several layers
- Softmax layer [41]: output layer used for caterogical classification training

In the present work, only the Softmax layer will be used for training in formal domain.

2.1.2 Error gradient backpropagation algorithm

In both Artificial Neural Networks and their biologic inspiration, the information is not located inside neurons, but rather inside the connection between neurons: the so called synapses. Indeed, those connections are endowed with synaptic weights, a coefficient that modulate the sign and intensity of the information flowing through. The knowledge contained in the network is therefore represented by the distribution of those weights among synapses. The whole goal of Machine Learning is to find the weight distribution so that the neural network behavior fits the desired function. In a small topology such as LeNet5 [37] (see 2.3), there are 369174 synaptic weights, and this number is exploding when addressing state-of-the-art CNN topologies, such as VGG-16 [28] which implies 138 millions parameters or even Transformer networks [29] that can reach billions. Understandably, an automatic learning algorithm is used to tune those weights. The most common family of training algorithms is based on Error Gradient Backpropagation [42] [26] [43]. The basic idea of this process is to learn statistical biases by iterating on large labelled training datasets. To do so, the learning algorithm iterates across training samples, starting with a random weight distribution. Each sample is associated to a label, which represents the target output of the network for this data. The data is passed at the input of the network, and the output is retrieved: this is the forward path. An error is computed between the actual output and the expected one (computed from a label or a specific ground truth value), using various kind of loss functions. At this stage, one is able to tune the weights of the last layer in order to minimize this error. The error is then propagated backwards, layer-by-layer, from the output towards the input. At each layer, the weights are automatically tuned to minimize the layer-wise error. More information on this process, including the mathematical principles and equations are presented in [42]. However, as we do not study learning algorithm, such details is out of scope of this work.

2.2 FNN in hardware

Now that the principles of Formal Neural Networks have been explained, this section will focus on the state-of-the-art regarding the deployment of such models on hardware platforms. It should be noted that in this whole work, we focus on neural network inference and training is considered out of scope: it is always performed in software, and the hardware platforms only come into consideration for the inference issue. At this stage, there are two major possibilities for hardware artificial neural network implementation: digital or analog systems. Digital implementations have 3 major advantages over analog circuits:

- The development and deployment frameworks are more mature and accessible for digital systems.
- The digital paradigm enable multiplexing, sequencing etc. which brings scalability.
- Analog circuits are more subject to electrical noise and technology variability, even more when scaled-up to larger system sizes.

Moreover, if analog circuits are usually more compact than equivalent digital circuits, this trend reverses when addressing smaller than $22nm$ CMOS technology [44]. Following Moore's law, the $22nm$ and smaller CMOS technology is democratizing fast, so digital circuits also take the advantage regarding compactness. For all those reasons, we only address digital implementations in this work. Digital systems comes in 4 main families:

- Central Processing Units (CPUs): the most generic and common type of programmable processors, used in every computers, smartphones, and general purpose electronic devices. This is a purely centralized Von Neumann type of architecture [45], were computation is highly sequential in a fast and generic core, which communicates with external memories.
- Graphical Processing Units (GPUs) [46]: a more specific type of accelerator, originally used for graphics rendering in video-games or special effets softwares. GPUs are highly parallel programmable processors, composed of a very large number of small and simple cores, dividing large computations into small portions.
- Application Specific Integrated Circuits (ASICs) [47]: a type of accelerator which is specifically designed for a given task, offering an optimal use of hardware resources (area, power, energy...) on this task. However, this type of accelerator is very expensive to produce, and is only conceivable for large production scales.
- Field-Programmable Gate Arrays (FPGAs) [48]: this type of device is composed of an array of processing elements (Look-Up-Tables, Registers, Multiplexers, Digital Signal Processing units...), which can be configured to build complex circuits using an hardware synthesis software. The architecture is described using a Hardware Description Language [49] (VHDL [50], Verilog [51]...), and automatically mapped to the device. The resulting accelerator can be highly specific and thus makes a near-optimal use of resources for a given task. FPGAs are way cheaper than ASICs to exploit, but make a less optimal use of resources, considering all the logic used to program the device, and the unused elements of the board. Moreover, FPGAs are re-programmable, which makes this technology well suited for early prototyping and research.

In the next section, a literature review will help us compare hardware neural networks on those 4 types of digital hardware targets.

2.2.1 Hardware neural networks on digital hardware

In [2], the authors presented the recent advances in the field of Neural Network acceleration regarding compression, algorithm optimization and hardware optimizations. The authors also proposed a separation of hardware acceleration in three levels: structure level, algorithm level and implementation level. The structure level deals with optimizations related to the network topology, such as synapse pruning (removing synapses with insignificant weights), weight redundancy and

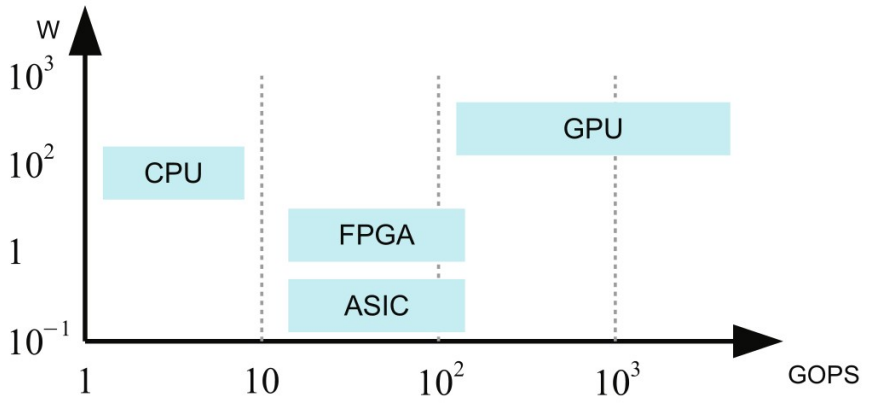


Figure 2.4: Summary of comparisons between CPU, GPU, ASIC and FPGA for neural network acceleration in terms of power consumption and computing power [2].

layer decomposition. The algorithm level deals with efficient algorithms for training, and are thus considered out of scope. Finally, the implementation level deals with hardware implementation platforms and techniques for CPU, GPU, FPGA, ASIC and other novel technologies such as memristor-based neural networks [52]. Concerning this implementation level, the authors made a comparison of various CNN topologies on various hardware targets, in order to measure and compare the subsequent efficiencies. Their results are summed-up in Figure 2.4, which shows the power consumption and computing power (GOPS: Giga Operation Per Second) of various neural networks on various hardware targets. In this figure, we can see that CPU is the least optimal hardware, offering low computing power for high power consumption. On the other hand GPUs offer the highest possible computing power, but with the highest power consumption. Finally, ASICs and FPGAs seem a good trade-off between computing power and power consumption, as they are specifically designed to optimize both these aspects. ASICs still offers lower power consumption, as it is even more specific than FPGA and does not imply programming logic and unused elements.

In [3], the authors also proposed an in-depth comparison of Neural Network acceleration regarding speed and performance per Watt, using various CNN topologies and hardware targets. The results of their study is given in Figure 2.5. If their study mainly deals with binarized neural networks, they also proposed results for non-binarized models. On those graphics, one might see that, for non-binary models, CPU and GPUs have quite similar performances, with a slight advantage for the CPU in both metrics. On the other hand, FPGA and ASIC architectures always offer better performance and speed-up than CPU and GPUs, in all configurations. This result is an interesting preliminar justification for SNN that belongs to also use binary coding whose activations are scheduled in time.

In [4], Nurvitadhi *et.al.* performed an in depth comparison on Neural Network acceleration targeting FPGA and GPU, regarding performance and performance per Watt. The authors also evaluated the influence of pruning, data compacity and binarization on the performance on both hardware. To do so, the authors compared various Deep Neural Networks (DNN) topologies on Nvidia Titan X, the most powerful GPU on the market, with Xilinx Stratix 10, the latest version of large Xilinx FPGA device. The authors also evaluated Xilinx Arria 10, a smaller recent FPGA device. The authors also developed a FPGA-based DNN accelerator architecture, which they used

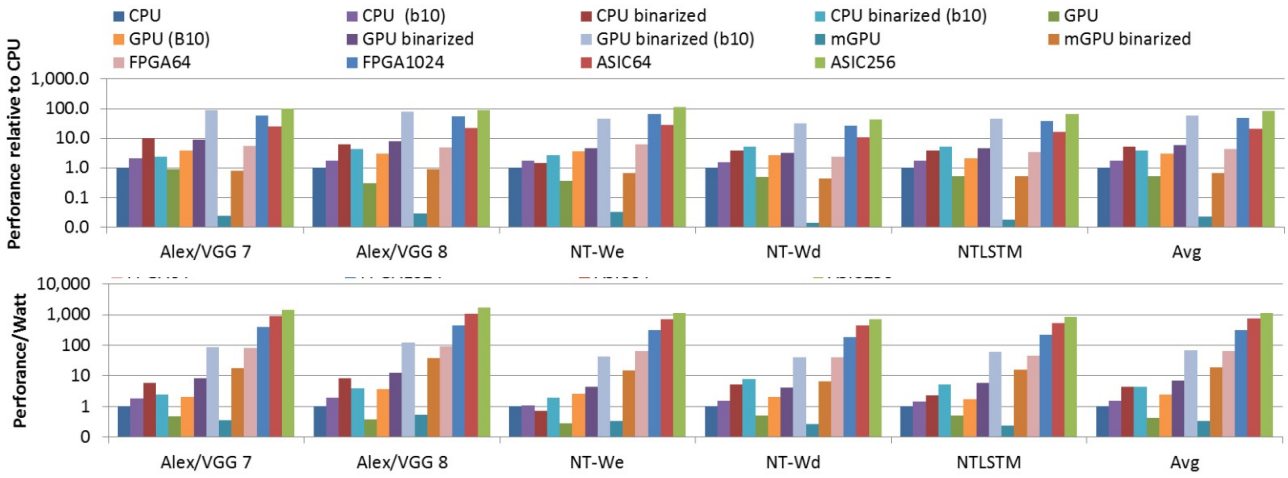


Figure 2.5: Comparison of CPU, GPU, ASIC and FPGA for binary neural network acceleration in terms of speed and performance per Watt, on various neural network topologies [3].

in their comparison. The results of the comparison are shown in Figure 2.6, for various baseline and several optimizations: pruning, compact data and binarization. Concerning raw performance, the only case where Stratix 10 FPGA is below the TITAN X is for baseline DNN. With optimizations, the FPGA outperforms the GPU in all cases. Concerning performance per Watt (*i.e.* energy efficiency), the FPGA performs better than the GPU in every experiment. Finally, the authors conclude that Neural Network acceleration should focus on FPGAs rather than GPUs, in the light of their measurements. However, most of these comparisons do not take into account the impact of compression on network accuracy. This metric has to be considered in addition to the speed and performance.

2.2.2 Example of FNN accelerators for FPGA

In the light of this literature review, it is clear that specific circuits such as FPGAs and ASICs are the most suited to Neural Network acceleration in the context of energy-and-resource-constrained embedded systems. Although there is a slight advantage for ASICs in that regard, FPGA's reconfigurability offers interesting prototyping capabilities, for a much lower cost than ASICs. Hence, in our work, we focus on FPGA-based neuromorphic accelerators. In this section, we will specifically describe two FPGA accelerators for Formal Neural Networks: VGT, which is an architecture for fully parallel FNN acceleration, and C-HLS, which is a fully-sequential FNN accelerator. Those two accelerators will be used in our application benchmark in Chapter 5. For information, a wide survey of existing formal accelerators is proposed in [53].

2.2.2.1 VGT accelerator

VGT (VHDL Generation Tool) [5] is a framework for Formal Neural Network deployment, which automatically generates VHDL code from an high-level description. In this work, the generated circuits are referred to as VGT architectures. It should be noted that the subsequent VGT accelerators is implemented with a high level of parallelism. Convolution layers operate in a

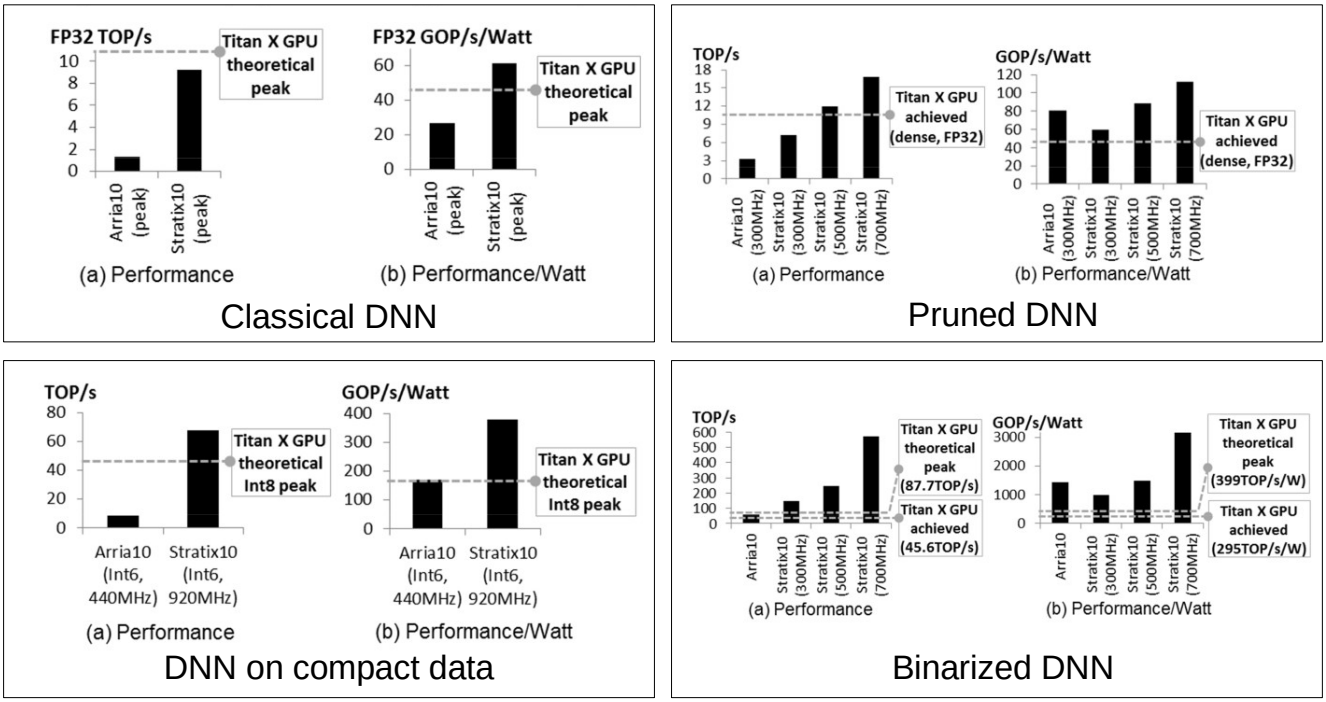


Figure 2.6: Comparison of FPGA (Stratix 10) and GPU (Titan X) for Deep Neural Networks in terms of performance and performance per Watt, for various optimizations: pruning, compact data and binarization. [4].

streaming fashion: the input feature-maps are flattened, and streamed pixel by pixel to the layer, in parallel for each input channels. In the same way, the output feature-maps are streamed in parallel one pixel at a time. The architecture of the convolution itself is made of two stages: a Processing Element for weight-multiplication (*i.e.* convolution of input channels with filters), and an adder-tree for filter-wise integration (*i.e.* sum of weighted inputs to compute output feature-maps). The full architecture is available in Figure 2.7. There is a Processing Element for each input channel. Each Processing Element operates weight multiplications in parallel for each convolution filter. Using a sliding window, weight multiplications are performed in a pipelined systolic fashion [10]. An illustration of a Processing Element architecture is available in Figure 2.8. After the systolic arrays, an adder-tree is used to sum the results filter-wise, so that there is one output value per output-filter. As the input feature maps are streamed to the PEs, and thanks to the pipelined architecture, the convolution layer will produce one output value at a time for each filter, in a streaming fashion.

The max-pooling layers of VGT works similarly to the convolution layers, except that there is no weight multiplication, and that the adder-tree is replaced with comparator-tree in order to extract the maximum value of each input patch. An illustration for the max-pooling layer is available in Figure 2.9

Finally, the fully-connected layer of VGT is implemented as a parallel matrix-multiplication architecture. The architecture is similar to that of the convolution layer depicted in 2.7. In contrast with Convolution and Pooling layers, the input of the FC layer is not streamed but available all in one piece. This is possible thanks to the flattening layer, that will not be described here. The FC layer itself is made, as for Convolution layers, of a PE stage and an adder-tree stage. In the

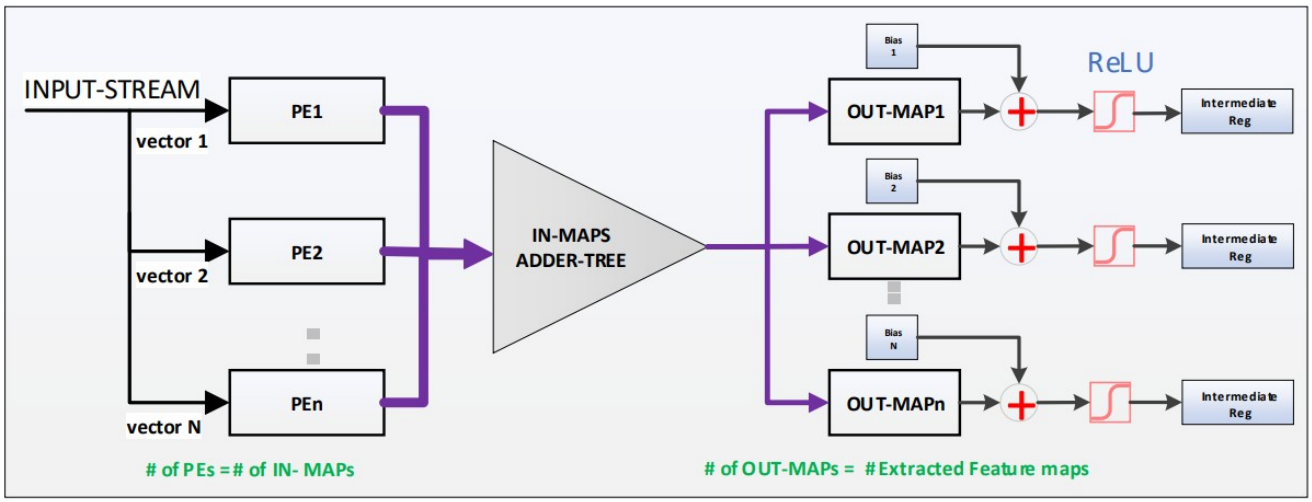


Figure 2.7: Illustration of a VGT-generated convolution layer architecture [5].

PE stage, there is one PE per input channel. Those PE stages perform weight multiplications in a fully-parallel way: all input activations are multiplied by all synaptic weights in parallel, thus for a 100-100 layer, there are 100×100 parallel multiplications. Then, the results are summed-up to obtain the output activations, using pipelined adder-trees. There is one adder-tree per output neuron so all output activations are processed in parallel.

2.2.2.2 C-HLS accelerator

The second FNN accelerator, which we call C-HLS accelerator, has been developped by Sebastien Bilavarn in the LEAT laboratory. This architecture is derived from a C code completely designed to support HLS. This code is a layer based implementation allowing 2D convolutions, ReLU, max pooling and fully connected layers. Layers are configurable (size, input, kernels, ...) such that typical neural network topologies like LeNet can be fully specified and simulated. Therefore this code can also be used to generate RTL IP cores that can be further integrated with a CPU and system bus to be quickly ran on Xilinx devices (Vivado HLS and SDSoC methodology). Loop level parallelism can be explored introducing pipelining pragmas in the original designs, but this is not adressed in this study, where this accelerator is parameterized in a fully-sequential fashion. This choice has been made in order to enhance the comparison between sequential and parallel accelerators in terms of hardware footprint and performance in Chapter 5.

2.3 Spiking Neural Networks

In previous section, we mainly focused on "hardware-level" optimizations. If we refer to the terminology proposed by [2] (introduced in section 2.2.1), there are other possibilities for optimization at algorithmic and structural level. In this PhD thesis, we also study the algorithmic-level optimizations by addressing a brain-inspired type of neuron: spiking neurons. Consequently, in this work, we investigate how a neural coding paradigm shift towards spiking domain could help mitigating logic-resources occupation, power and energy consumption of FPGA accelerators. Of course, this matter is always addressed with hardware considerations in mind: the aim is to build an hardware

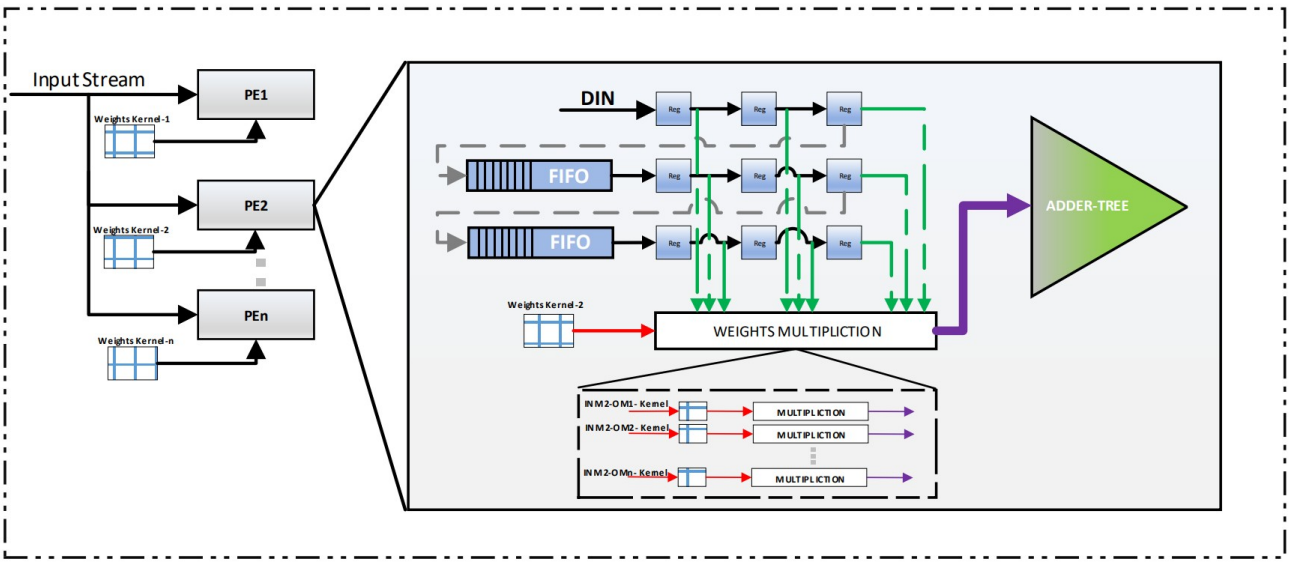


Figure 2.8: Illustration of a Processing Element in VGT convolution layers [5].

platform which takes advantages of the model, and find a model which takes advantages of the hardware possibilities. But first, a short review of Spiking Neural Networks principles will be given to introduce all the concepts involved in our study. Then, hardware implementations of Spiking Neural Networks will be discussed.

2.3.1 Principle of Spiking Neural Networks

Spiking Neural Network is a type of brain-inspired Neural Network that emerged from the neurosciences, where it aimed in emulating the biological brain behavior for scientist to study and understand the physical mechanisms underlying information propagation in biological neural networks. The first model to emerge was the very simple Integrate and Fire (IF) neuron model proposed by Lapicque in 1907 [54]. Later, more biologically plausible neuron models were proposed such has the Hodgkin-Huxley (HH) [55] neuron which used a vast set of non-linear differential equations in order to model the mechanism of action potential initiation and propagation in Giand Squid axons. Figure 2.10 shows the biological plausibility of various spiking neurons (*i.e.* how close they mimick the real physical mechanisms occurring in the biological brain) against their implementation cost in terms of Floating-Point Operation Per Seconds (FLOPS). As it can be seen in this representation, the more a model is biologically plausible, the higher its implementation cost. Thus the HH neurons, and other complex models are not well suited to our search for resource and power efficiency in Machine Learning tasks. For this purpose, we tend to look for models with the lowest implementation cost possible, which we found in the IF neuron model. Moreover, it has been shown by Brette *et. al.* [56] that the IF-based neuron models are in fact very realistic models when modeling spiking activity in the brain, and are widely used in the literature for that purpose [57] [58]. As this model is both hardware-friendly and effective, we mostly use IF neurons in our works.

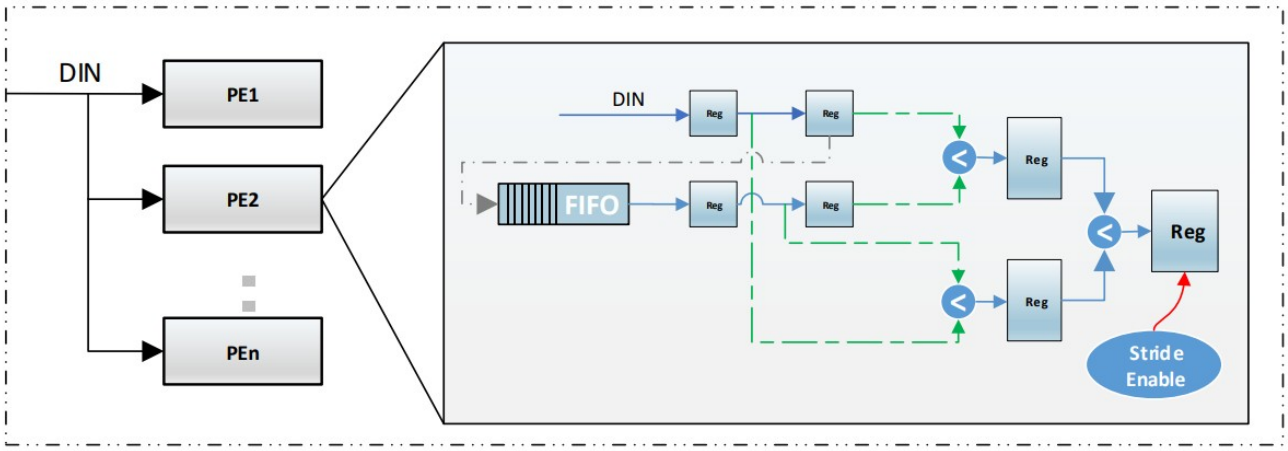


Figure 2.9: Illustration of a VGT-generated max-pooling layer architecture [5].

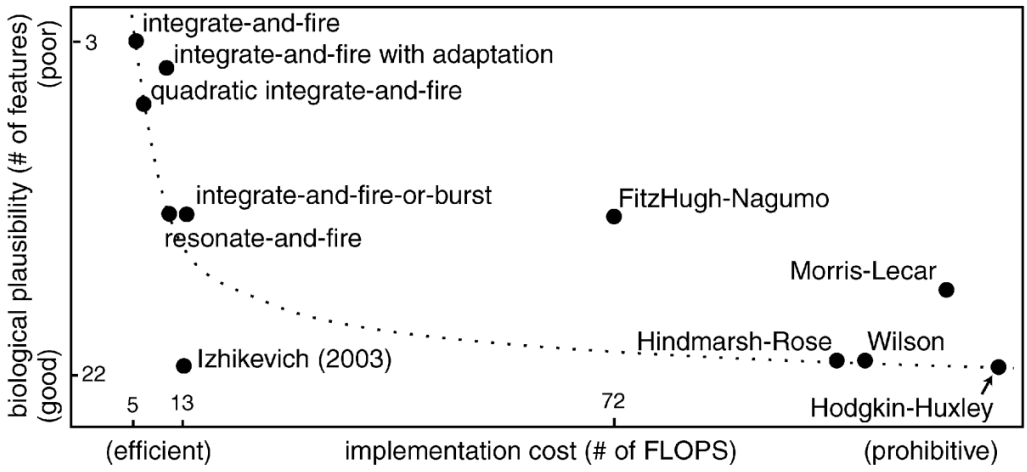


Figure 2.10: Biological plausibility of Spiking Neuron models against implementation as stated in [6]

The IF neuron dynamics are governed by the differential equation 2.2.

$$\begin{aligned} \frac{dU_i^l(t)}{dt} &= I_i^l(t) - S_i^l(t) \times \theta \\ S_i^l(t) &= \Theta(U_i^l(t) - \theta) \end{aligned} \tag{2.2}$$

Where U_i^l is the membrane potential of neuron i of layer l, S_i^l the state of the axon (output synapse) of neuron i, θ is the neuron threshold and Θ is the Heaviside step function. Moreover, R is the input resistance of the neuron and I_i^l is the input current generated by the input spikes as shown in Equation 2.3:

$$I_i^l(t) = \sum_j w_{i,j} \times S_j^{l-1}(t) \tag{2.3}$$

Where $S_j^{l-1}(t)$ is the input spike train and $w_{i,j}$ are the input synaptic weights. A common approach

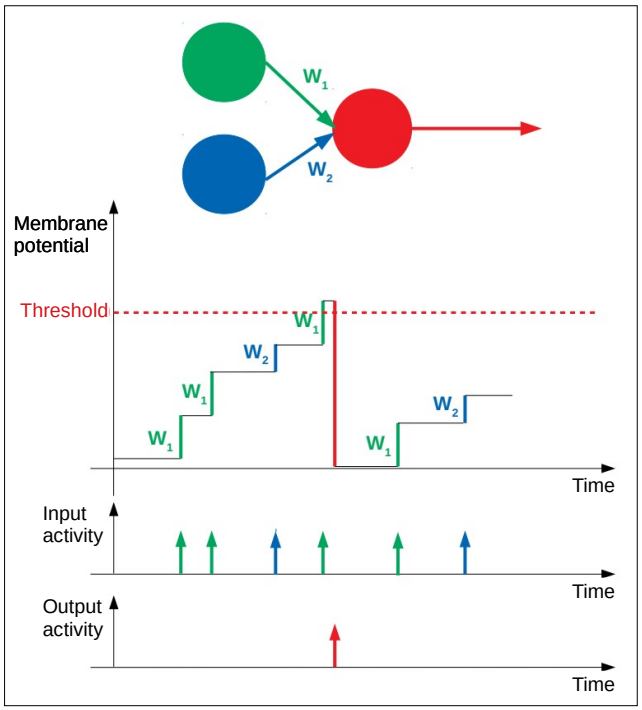


Figure 2.11: Illustration of the Integrate & Fire neuron model. Top: network setup, bottom: membrane potential, input and output activity.

is to approximate the solution of the differential equation 2.2 using the Euler method. The discrete solution is shown in Equation 2.4:

$$U_i^l[t+1] = U_i^l[t] + \sum_j w_{i,j} \times S_j^{l-1}[t] - S_i^l[t] \times \theta \tag{2.4}$$

In other words, the membrane potential is incremented by the sum of the weighted input activations (*i.e.* input weights since input activations are binary), and decremented by θ whenever the neuron threshold is exceeded. Additionally, over-passing the threshold triggers an output spike ($S_i^l[t] = 1$). This process is summarized in Figure 2.11. The neuron setup depicted in the figure features a red neuron receiving input spikes from the blue and green neurons. The membrane potential graph shows the impact of weighted input spikes on the red neuron potential. The bottom graphs show the input and output spikes from the point of view of the red neuron. As shown in this Figure, the IF neuron integrates temporal and spatial information and produces an output which is propagated towards the following neurons. Thus, its behavior is not far different to that of a Formal neuron, except that i) multiplication-accumulation (MAC) operations are replaced by accumulation operations (ACC), and ii) the model is event-based: spikes are received and processed asynchronously. Hence, SNN acts as a subset of binary networks whose activations are scheduled in time.

Those IF neurons are used to replace standard Formal neurons in mostly any kind of Neural Network Topology. Thus, their exits Spiking Multi-Layer Perceptrons [59], Spiking Convolutional Neural Networks [58], Spiking Auto-Encoders [60]... The only difference is the model of neuron and the encoding of information in the network. In all, SNN process information in a dynamical

fashion using binary activations, whereas FNNs process is static but uses real-valued activations.

2.3.2 Spike encoding

If an FNN works on classical data (static images, videos...), SNNs work with spikes. Thus, the network's input data must be in event-based format. There are two ways to obtain such event-based data: either by using a specific event-based sensor, or by transcoding classical data toward spiking domain.

Event-based sensors are mostly present in the field of vision sensors, with the recent breakthrough of Event-Based Camera, also called Dynamic Vision Sensors or Artificial Retinas [61]. This type of sensor is directly inspired from the biological retina behavior: it is made of an array of sensors, each of which detects intensity gradients and generate an event whenever the measured gradient exceeds a certain threshold. Those events usually follow the Adress Event Representation (AER) policy, in which information is represented by a packet containing locations and timestamps of events. The generated event flux then represent the motion in an efficient way, as only the moving objects generate events, and not the static background. Other type of event-based sensor exist for other type of data, such as artificial cochleas for audio data [62], or even artificial event-based skins for tactile sensors [63]. Thanks to the AER representation, event-based sensors produce less information than their classical counterpart, as most of pixels are idle. Moreover, the data itself is lightweight, as it only encodes position and timestamp. Moreover, such data can be directly interpreted by an SNN, and doesn't require a dedicated transcoding pre-processing.

However, event-based sensing is a novel technology which is not yet widespread in industry. In most use-cases, only conventional frame-based sensors are available. Such applications thus require a dedicate pre-processing to encode data into spikes. Their are 3 main spike encoding policies explained below. It should be noted that in our explanations, we deal with images and pixels, but all the notions are translatable towards other type of data.

- Rate coding [64], which consists in generating spike trains for each input pixel whose frequency are proportional to input data;
- Latency coding [65], which consists in generating a single spike per input pixel, where the emission date is proportional to the pixel's intensity;
- Order coding [66] [67], which consists in emitting one spike per pixel in the order of intensity. Pixels are sorted by intensity: the brightest pixel spike first, followed by the second brightest, and so on.

Those three different methods are illustrated in Figure 2.12 for image transcoding case. The pixel intensity is represented by grayscale squares, and the generated spikes by the red arrows on the time axis. The rate coding approach usually offers the best performance, but generates a high number of spikes that drastically impacts both the computation density of the processing and its latency [64]. The latency coding [68] offers inferior performances, but enables to drastically reduce the computation density by reducing the number of spikes flowing into the network. However, the computation duration remains high, due to the width of the time-window [64]. Finally, the order coding seems the most promising: it reduces performance but also reduces both the event density and the computation duration by compressing the spike emission window [66]. However, rate-coding was used during most of this thesis. At the time this work started, that was indeed the most reliable spike encoding policy in terms of network accuracy. Moreover, rate-coding is

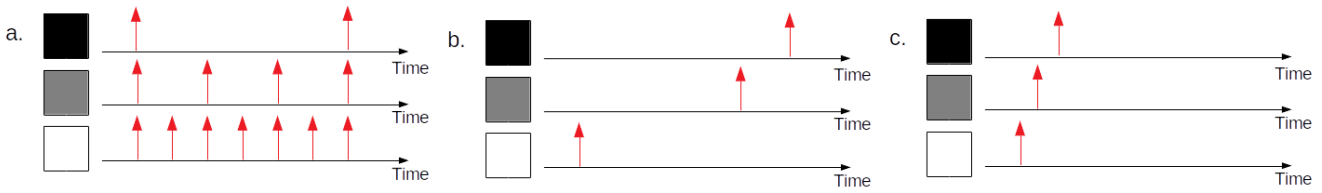


Figure 2.12: Different spike coding methods: a. is rate coding, b. is latency coding, c. is order coding.

widespread enough so that it is well documented and supported in most SNN frameworks. The rate-coding formula used in this thesis is given in Equation 2.5.

$$\begin{aligned} \mathrm{P} &= \frac{1}{\mathrm{MaxFreq} + (\mathrm{MinFreq} - \mathrm{MaxFreq}) \times v}, \\ v &= \frac{1 - i}{255} \end{aligned} \tag{2.5}$$

Where P is the spike train period associated to a pixel of value i (8-bit integer), MaxFreq and MinFreq are respectively the maximum and minimum spike train frequencies.

It should be noted that even if most of this work is based on such rate-coded SNNs, other types of encoding where studied during the thesis. In Chapter 7, we address timestep-constrained SNNs trained through Surrogate Gradient Learning [32] [69]. This type of SNN drastically differs from the rate-coded representation.

2.3.3 Training SNNs

In this subsection, training techniques for Spiking Neural Networks will be presented. It should be noted that in this work, we focus on classification tasks. If FNNs mostly use variants of the well-known Backpropagation algorithm [25] [26], in SNNs the learning phase can be performed by three very different techniques:

1. Neural network conversion
2. Spiking Time Dependant Plasticity (STDP)
3. Spiking Backpropagation

The first training technique, and the most widely used in the literature, is neural network conversion [70] [57]. This method consists in training a neural network using ReLU neurons in Formal domain using conventional Backpropagation algorithm, and then transferring the learned synaptic weights to a Spiking Neural Network with the exact same topology. Some attention has to be given to the compatibility of the trained network with SNNs: for example, there ar no mature ways of implementing Average Pooling layers in SNNs, thus we would rather use Max Pooling layers. Similarly, batch normalization techniques are difficult to apply in SNNs, thus it should not be used in training. Conversion is simple to set, and offers state-of the art classification performance in spiking domain. It is compatible with the vast majority of SNN deployment frameworks and neuromorphic accelerators, thus this is the method we have chosen to use in our work.

Secondly, Spiking Time Dependant Plasticity (STDP) [71] [72] is an unsupervised and on-line training method which exploits the causality effect between spikes to perform data clustering. On-line means that the training phase is not properly distinct from the test phase, as the network is continuously learning.. Unsupervised means the training data is unlabeled: *i.e.* it does not require human effort to label the training set. Indeed, the clustering relies on dataset intrinsic statistics, and not on human-chosen classes. Additionally, STDP is a local learning rule: each synaptic weight is tuned by the relative spiking times of uphill and downhill neurons. If an input spike causes an output spikes (*i.e.* if an output spike is generated shortly after receiving an input spike), the synapse is potentiated. On the other hand, when an input spike does not cause an output spike, the connection is depreciated. This local aspect makes STDP more hardware-friendly than traditional Backpropagation. Moreover, the computation itself is much simpler, as it does not require complex partial derivative computation. However, this training method is very recent and still under development. Notably, STDP does not provide state-of-the-art performance on most classification tasks [73].

The third training method is Spiking Backpropagation [74] [75], also called Surrogate Gradient Learning [32]. This method is an adaptation of the traditional Error Gradient Backpropagation algorithm to Integrate & Fire spiking neurons. Indeed, in Error Gradient Backpropagation, the weight increment (increase or decrease) is proportional to the derivative of the activation function. However, as we have seen in section 2.3.1, the activation function of the IF neuron is a simple threshold, *i.e.* an Heaviside function. The point is that first, this function is not differentiable in 0, and more importantly, that its derivative equals to 0 elsewhere, thus implying a null weight update. In order to get around this limitation and still apply Backpropagation to networks of IF neurons, the creators of Spiking Backpropagation used an approximation of the Heaviside function (usually a sigmoïd or hyperbolic-tangent-based function) to perform the backward path, while the forward path uses the normal Heaviside activation function. This method has the advantages of offering state-of-the-art classification performance on most datasets, all the while using fewer spikes and inference time than converted SNNs [73]. Most of the thesis is based on rate-coded converted SNNs, but Spiking Backpropagation has recently gained visibility and so have its promises for hardware acceleration. Hence, Chapter 7 addresses the potential benefits and hardware implementation of SNNs trained via Surrogated Gradient Learning.

2.3.4 Terminate Delta

It has been shown in section 2.3.2. how to encode conventional data like images and time series into spikes. In this subsection, the Terminate Delta process is explained. It is used in this work to interpret the output spikes and retrieve the predicted class accordingly. This classification procedure is taken from N2D2 [1], a Neural Network training and deployment framework which enables conversion towards spiking domain.

This process consists in determining the most active neuron, using a margin whose characterized by the parameter Delta. With a Δ value of 2, it means that the most spiking neuron would be enacted after it has spiked two times more than the second most active neuron. Indeed, with rate-based SNNs, spikes arrive frequently at the output. Hence, tuning Δ enables to take account of more or less spikes before enacting the result. A high Δ value will give better classification performances, but higher execution time and higher number of spiking events in the network. Thus, the Delta parameter is an interesting parameter to tune the trade-off duration and energy against accuracy.

2.4 SNNs in hardware

Spiking domain is expected to bring drastic hardware footprint reduction to neural network accelerators compared to conventional formal coding domain. Indeed, the simplicity of the Integrate & Fire (IF) neuron model, coupled with the lightweight binary activation, could bring both logic resource reduction and power efficiency.

2.4.1 Advantages of SNNs in Hardware

Spiking Neural Networks have several advantages over Formal Neural Networks when addressing hardware implementations. The first lies in the activation integration mechanism: the synaptic operation. For Integrate & Fire spiking neurons, as described in Equations 2.4, the integration consists in accumulating a synaptic weight whenever a spike is received. Basically, this consists in a simple Accumulation (ACC) operation. For formal neurons on the other hand, the synaptic operation is a Multiplication-And-Accumulation (MAC), as the weight is multiplied by the real-valued activation. This difference has two main impacts on FPGA implementations: the MAC uses a Digital Signal Processing (DSP) unit, which is a scarce resource on most FPGAs. Moreover, the MAC operation is considered to be at least ten times more energy-intensive than the ACC operation. Indeed, on 45nm CMOS, Panda *et. al.* [76] reported $3.2pJ$ for a 32-bit MAC against $0.1pJ$ for a 32-bit ACC. The ratio of energy consumption between MAC and ACC operation will be studied further in Chapter 4.

Consequently, hardware SNNs should be more energy-efficient than FNNs [31] [77] [78], based on the energy cost of their respective synaptic operations. Additionally, spikes can be represented by binary (1-bit) signals, which offers lightweight connections between neuron, where formal activations are encoded on signal from 8 to 64 bits. This enables better scalability for parallel implementations, which involves a large number of physical connections between hardware neurons [79]. Finally, the event-based processing also offers interesting properties for highly-constrained embedded systems. Indeed, in SNNs, activations are scattered in time in a sparse fashion, whereas all synapses of a layer are activated simultaneously in FNNs. In other words, the information in SNNs is dynamic: it is spread on a temporal dimension, which offers low instantaneous computing density [79]. In an FNN however, the behavior is fully static, implying high computation density. This difference enables for better multiplexing possibilities, as well as low-power implementations. However, this low instantaneous dynamic power might be counterbalanced by longer processing time, thus energy consumption might not follow the same trend. However, energy is not the only constraint, and power is often a limiting factor in embedded-systems, in solar-powered systems for example.

2.4.2 Literature review

In this section, we give a short list of recent works in the field of FPGA SNN accelerators. For the architectures where the information was available, logic resources, execution time, power and energy consumption are summarized in Table 2.1. As some architectures are not named by their authors, we refer to them with names inspired from their respective article titles.

Table 2.1: Logic resources, execution time, power and energy of FPGA SNN accelerators found in the literature. Topology nomenclature: KcWsX = Convolution layer with K filters of size W^2 and a stride of X, KpWsX = Pooling layer with K filters of size W^2 and a stride of X. Fully-connected layers are referred to by their number of output neurons.

Name	Year	Data	Topology	# Synapses	Ressources	Power (mW)	Time per image (ms)	Energy per image (mJ)
"ConfConvNode" [7]	2018	events	CNN: 32x32-6c5s1-6p2s2-4c5s1-4p2s2-8c5s1-4c2s1	904	21K LUTs 38K FFs 6.5 Mb BRAM	7.7	25	0.2 (220 nJ/syn)
"Large Scale SNN" [80]	2012	events	Toroidal: 2048 neurons	2,048M	80K LUTs 91K FFs 20 DSPs 8.6 Mb BRAM	?	?	?
"Polychronous SNN" [8]	2013	events	Unstructured 4096 neurons	1,150M	135K LUTs 581K FFs 14.4 Mb BRAM	?	?	?
"Low Power SNN" [9]	2020	frames	CNN: 28x28-64c5s1-64p2s2-64c5s1-64p2s2-128-10	37,248	140K LUTs 81K FFs 16.5 Mb BRAM	4600	6.1	35 (940 nJ/syn)
"IIR SNN" [11]	2020	frames	MLP: 28x28-500-500-10	647,000	125K LUTs 185K FFs 1028 DSPs 3.1 Mb BRAM	4500	0.52	2 (3,1 nJ/syn)
HFirst [12]	2015	events	CNN: 128x128x1-12c7s2-12p4s1-36c8s2-36p32s1	2892	17 DSP 6 Mb BRAM	250	0.002	0.5 (170 nJ/syn)
S2N2 [13]	2021	time-series & frames	CNN: 16x16-64c5s1-64c5s1 128c3s1-128c3s1-1024-24	554368	102K LUTs 34K FFs 40 DSPs 12.7 Mb BRAM	?	0.031	?
SPLEAT [14]	2020	frames	CNN: 28x28-6c5s1-6p2s2-16c5s1-16p2s2-84-10	22894	4.8K LUTs 3.2K FFs 1 DSP 0.45 Mb BRAM	315	1,4	0,5 (22 nJ/syn)

2.4.2.1 "ConfConvNode"

In [7], Camunas-Mesa *et. al.* introduced a configurable event-driven architecture for convolutional layers targetting FPGAs. This architecture addressed IF neurons, and a conversion technique was used for training. The core of this architecture is a Processing Element (PE), a building block for user-defined spiking CNN architectures. For a better understanding, a schematic of the circuit of a PE is given in Figure 2.13. The PE involved a convolutional processing unit and a routing module. The convolution processing unit was used to convolve a spike with the convolution filters, and perform the IF neuron thresholding operation. The routing module was in charge of reading the input spike addresses and sending the output spikes to their destination neuron. In the paper, the architecture was used to classify poker cards filmed in real time ($25ms$ per sample) using a Dynamic Vision Sensor (DVS) [61] (see section 2.3.2). In real-time, the architecture used $7.7mW$ and $192.5\mu J$. The authors also evaluated the architecture on slowed DVS recordings: using a slow-down factor of 10, the architecture used $5.25mW$, and $0.85mW$ with a slow-down factor of 100.

2.4.2.2 "Large Scale SNN"

In [80], Cheung *et. al.* introduced a fully-parallel SNN hardware architecture targetting off-the-shelf FPGA-based systems. As this architecture primarily targeted cortical simulation, the implemented neuron was an Izhikevich model [6], but the authors claimed that it could be easily changed to a more hardware-friendly IF model. The design was highly-pipelined, ensuring low ex-

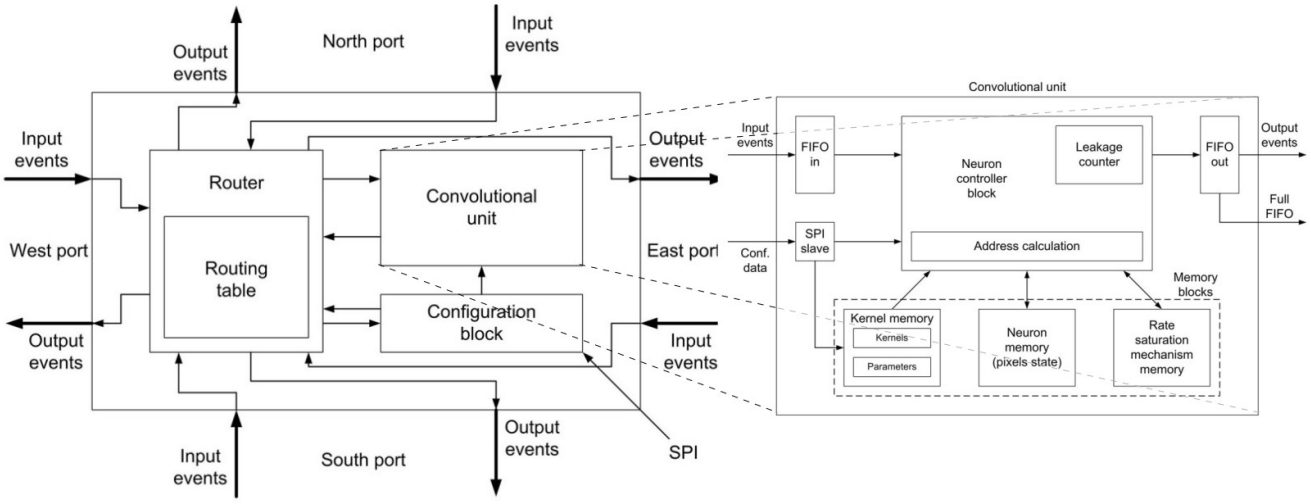


Figure 2.13: Illustration of the Convolutional Processing Element found in [7].

ecution time. Moreover, the architecture involved a weight-distribution module, which was used to parallelize accesses, distributions and integrations of synaptic weights concurrently. Additionally, the synaptic weights were arranged in consecutive RAM locations, enabling to retrieve several data in a single RAM access. This weight-distribution module is illustrated in figure 2.14. Moreover, the architecture worked in a fully event-driven way thanks to FIFO queues between neurons. This architecture supports 64000 neurons at a maximum rate of 1.39×10^9 spikes/s , with a spike-rate delivery up to 1.4 times faster than GPU at the time of the study (2012). However, this metric is not standard and not comparable to other architectures. Moreover, the number of spikes strongly depends on the data and spike encoding. The impact of data on number of spikes will be studied in details in Chapter 4.

2.4.2.3 "Polychronous SNN"

Reference [8] introduced a hardware architecture for polychronous Spiking Neural Networks [81]. This type of neurons is based on latency coding rather than rate coding: spikes are associated with precise latencies so that they arrive simultaneously to their destination neuron and cause it to fire, despite being emitted asynchronously. The neuron circuit is illustrated in figure 2.15. In this figure, timers are used to detect when pre-synaptic spikes arrive within 3ms of each other. It should be noted that these delay is programmable. The network was trained in hardware using a kind of STDP learning rule, adapted to programmable delay synapses. Using this approach, this type of SNN is able to process complex spatio-temporal patterns. The architecture was highly multiplexed to ease scalability. In the article, the authors implemented a 4000 neurons network with 1.15 million programmable delay synapses on FPGA, and achieved real-time simulation of cortical neurons. The authors also claim that their implementation is robust to noise from random input spikes. This approach is promising for hardware implementations of Machine Learning applications: encoding information in relative timing of events could increase the quantity of information conveyed by spikes while reducing their number. However, STDP rules usually provide lower accuracy than other methods, thus the interest of this technique remains to be proven on classification tasks.

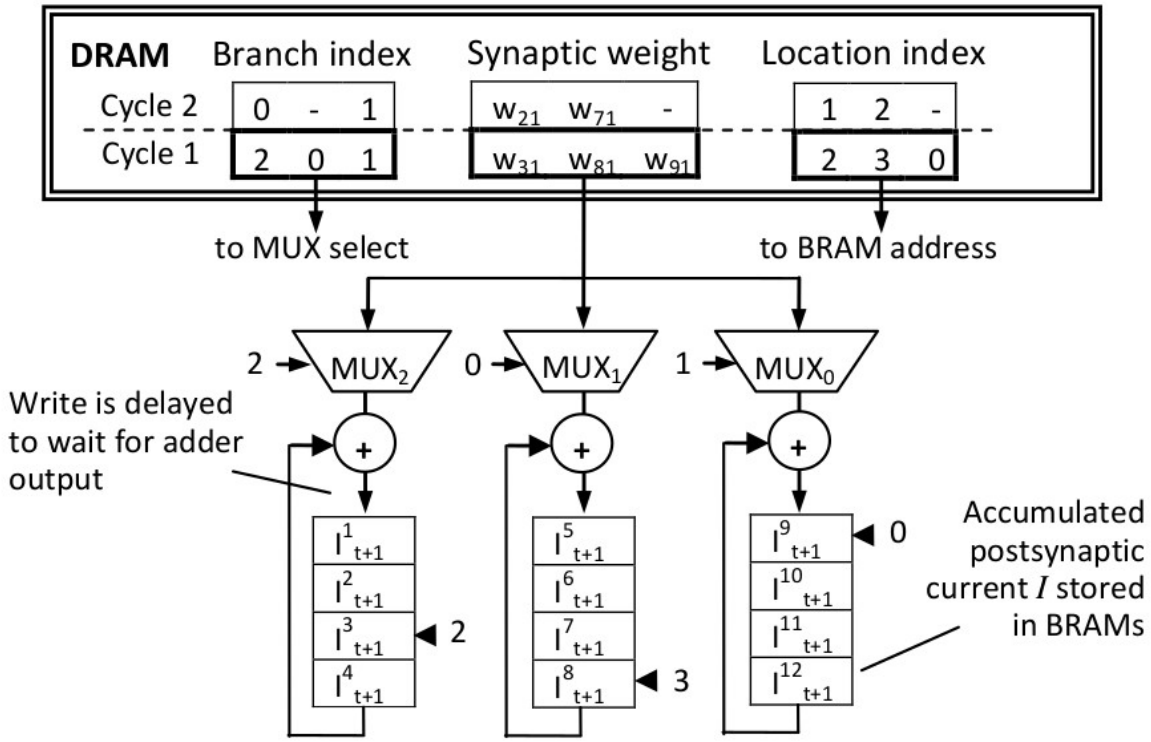


Figure 2.14: Illustration of the weight distribution technique used in [7] to limit the memory access rate. module has 3 input synapse data per cycle. The neuron 1 connects to neurons 2, 3, 7, 8 and 9 in the network, and neuron 1 is fired in the last time step. The figure shows the accumulation of three synaptic weights ($w_{3,1}$, $w_{8,1}$ and $w_{9,1}$).

2.4.2.4 "Low Power SNN"

In [9], Ju *et. al.* presented an SNN architecture targeting FPGA for integration in low-power systems, supporting all basic CNN layer types. The authors addressed the hardware-friendly IF neuron model, and used network conversion for training, with a weight balancing method [82] to optimize spike emission rate. Several optimizations were made to increase hardware efficiency of the architecture. First, authors used a 8-bit fixed-point dynamic. Moreover, convolutions were implemented in a parallel, pipelined and systolic fashion using shift-registers, ensuring fast processing. For a better understanding, a schematic representation of the pipelined parallel convolution implementation is shown in Figure 2.16. Additionally, the whole network was fully pipelined to further reduce processing latency. The architecture has been implemented on a Xilinx's ZCU102 FPGA, achieving 98.94% recognition rate on MNIST at 164 FPS. This is equivalent to an acceleration of 41× against CPU and 22× against GPU. The architecture used $4.6W$ and $35mJ$ per image. Additionally, the authors proposed an hardware-efficient implementation for spiking Pooling layers.

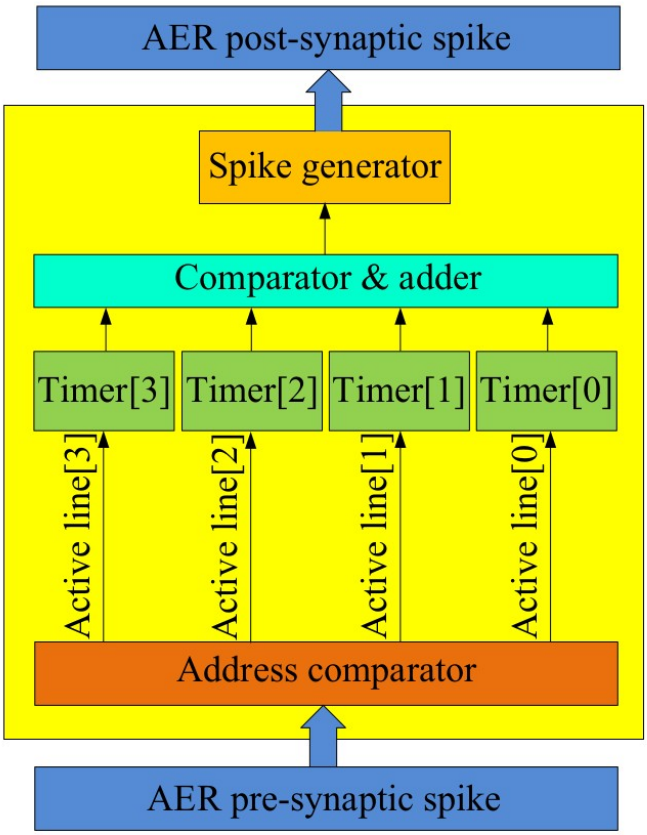


Figure 2.15: Polychronous neuron circuit overview found in [8].

2.4.2.5 "IIR SNN"

In [11], Fang *et. al.* proposed a novel way to implement for SNNs in hardware, and introduced their subsequent architecture. The accelerator supported all CNN standard layers. The authors used Leaky IF neurons, which they demonstrated to be equivalent to Time Encoding Machines. In the light of this demonstration, the authors developed a flexible model in which SNNs are emulated by a network of Infinite Impulse Response (IIR) units, which drastically reduce latency by enabling neurons to encode information on small time windows. In the architecture, each layer is emulated by a Processing Element in a multiplexed fashion, and it features a layer-level pipeline for execution time reduction. The whole architecture is generated through High Level Synthesis (HLS), as part of a global framework. This framework covers all steps from training, quantization, architecture generation and deployment. An illustration of the full framework is given in Figure 2.17. Training is performed thanks to surrogate gradient back-propagation (see section 2.3.3) using PyTorch. The framework involves resource optimization by using pragmas in the C-HLS code to increase or decrease parallelism according to the network size. Using their framework and architecture, the authors obtained up to 2.9× speedup compared to Intel's Loihi [83], 38.5× compared to SpiNNaker [84], 1.9× compared to TrueNorth [85] and 3.3× compared to Minitaur [86]. Concerning energy consumption per image, the authors achieved a 33.6× reduction compared to Nvidia RTX 5000, 2.6× compared to Loihi and 2.82× against SpiNNaker.

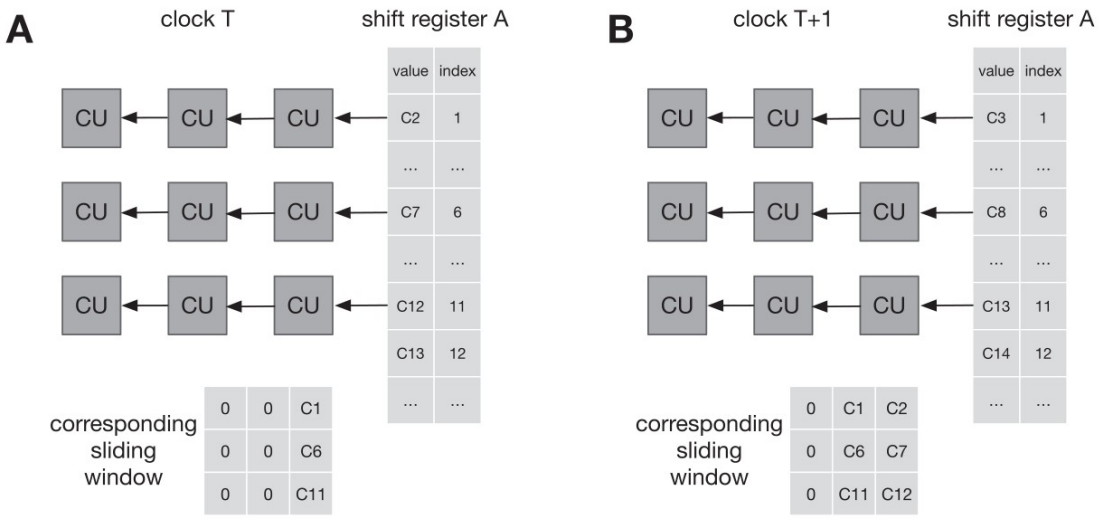


Figure 2.16: Illustration of the pipelined parallel convolution implementation found in [9]. The input data is first flattened in rows, and fed in a shift register. A systolic array of Convolution Units [10] then performs the operation in a pipelined fashion. A: Situation at clock T, B: situation at clock T+1.

2.4.2.6 HFirst

In [12], Orchard *et. al.* proposed an SNN architecture tailored for DVS signal processing, using Address Event Representation (AER), for spatial pattern recognition. The accelerator supported all standard CNN layers. According to the authors, the asynchronous nature of events ("frame free" data) frees computation and communication from the rigidity of conventional clocked systems. Using timing of spikes in object recognition, the authors demonstrated a drastic simplification of computation. Indeed, a simple asynchronous temporal-winner-take-all (temporal WTA) pooling operation was used instead of more complex synchronous operations usually implemented in SNN accelerators. This temporal WTA drastically mitigated the processing latency as there was no need to accumulate spikes to perform pooling. The FPGA accelerator was built in a highly parallel and pipelined fashion and was able to reach real-time recognition of DVS data such as written characters, written digits, and playing cards, with a latency inferior to $2\mu s$. Example of DVS recordings of such samples are shown in Figure 2.18. Additionally, the power consumption of the system depended on the nature of the scene. A scene without any movement used 100mW, but this value climbed up to 250mW for scenes with a lot of action (*i.e.* a lot of input events).

2.4.2.7 S2N2

S2N2 was introduced in [13], and was an FPGA architecture for streaming SNNs, which implemented Leaky IF neurons and time coding policy. The architecture supported all standard CNN layers. In this work, the term "streaming" was used to describe the temporal nature of data, which is streamed at the input of the network. The authors stated that SNN hardware implementation usually involved systolic array-based computation [10]. This kind of systems benefits from a supposed high sparsity in spiking events to reduce computational requirements. This work

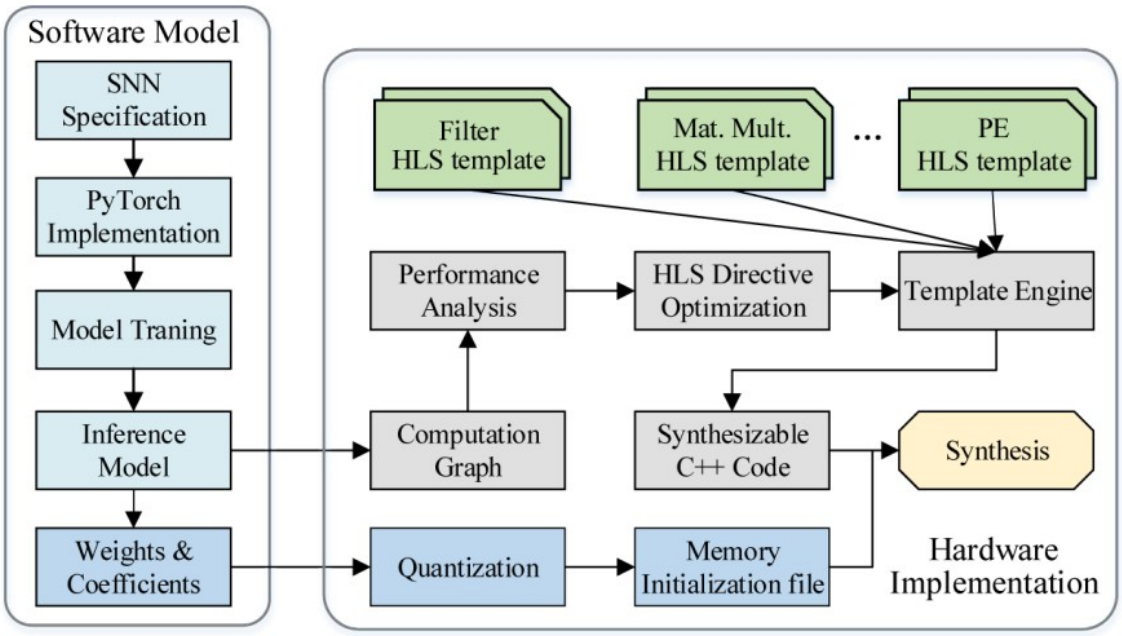


Figure 2.17: Illustration of the training, compression, conversion and hardware deployment framework proposed in [11].

showed however that this assumption was not always true in the case of data with large temporal dimensions, such as Deepsig RadioML 2018 dataset [18]. The authors proposed an architecture adapted to such streaming SNNs developed using FINN framework [87]: this framework is tailored for binary Neural Network, and has been customized by the authors to support SNN development and deployment. As it is shown in Figure 2.19, the architecture flattened input data into streams and convolution was operated in a parallel SIMD (Single Instruction on Multiple Data) fashion. The resulting architecture involved intra and inter-layer parallelism, and a pipeline to efficiently process data streams. The authors obtained a precision of 68.5% on RadioML 2018 dataset.

2.4.2.8 SPLEAT

SPLEAT [14] is an architecture for configurable SNN deployment on FPGA, which uses IF neurons. The architecture supports convolutional, pooling and fully-connected layers, and supports both temporal and rate coding policies thanks to a configurable spike generation cell. The architecture has been designed to be easily configured and deployed on FPGA. It features inter-layer parallelism (*i.e.* each layer is emulated by a dedicated Neural Processing Unit) and layer-wise pipeline to reduce execution time. For a better understanding, an illustration of the architecture of such Neural Processing Unit is given in Figure 2.20. Each Neural Processing Unit emulates a given layer in a highly sequential fashion, enabling high scalability. Due to this implementation choice, the FPGA logic occupation of this architecture is really low. On MNIST dataset, in average, the architecture used 315mW, with a an execution time of roughly 100ms per image. It should be noted that the processing time varies widely between two images, as the number of spikes vary from one sample to the other. The architecture also features a specific module for output classification, which interprets the spikes using a terminate delta process (see section 2.3.4).

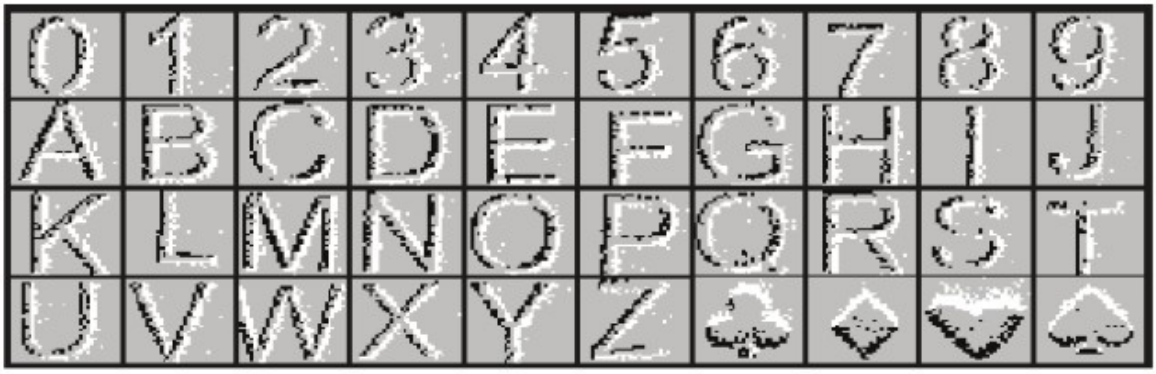


Figure 2.18: Illustration of stabilized DVS data used for training and testing in [12].

2.4.2.9 Conclusion

In this short literature review, we have seen that implementation choices vary greatly from one architecture to the other, depending on the chosen spiking neuron model and spike encoding policy. In all cases, much effort was given by the authors to reduce processing latency, by using pipelines and systolic architectures in some cases. Moreover, authors often developed specific hardware-friendly layers or neuron models in order to reduce hardware footprint of the subsequent implementations, in a software-and-hardware co-design fashion. Overall, this literature review shows that SNNs are promising for hardware footprint reduction of NN implementation on FPGA. However, this study lacks information on fair comparisons between hardware SNNs and FNNs. Therefore, this matter is specifically addressed in the next section.

2.4.3 Confronting Spiking and Formal Neural Networks

In this section, we introduce and describe previous literature works which tackle the comparison between formal and spiking neural network implementations on FPGA. In a first part, we address logic resources usage comparisons, and in a second part, energy consumption comparisons.

2.4.3.1 Logic resources comparison

In [88], Khacef *et. al.* compared a Multi-Layer Perceptron (MLP, *i.e.* a neural network made of several fully-connected layers) implementations on FPGA in both spiking and formal domains, in identical conditions. The SNN used IF neurons and rate-coding to translate static images into spike trains. The authors demonstrated a reduction of the FPGA logic resources occupation by 50% when using spikes compared to formal coding on MNIST dataset, showing that a single coding paradigm switch with no further optimizations could drastically reduce logic resources usage of NN implementations. Moreover, the authors showed a decrease of 59% in dynamic power usage, which is coherent with the difference in logic resources usage. However, this worked only compared the neuron layers, and did not take in account the resource overhead involved by spike encoding and output interpretation. Indeed, the spikes must be encoded into spike trains using rate-coding, which requires an additional encoding module in the FPGA. In a similar way, the output decoding via terminate delta process requires a dedicated module in FPGA. Thus, the

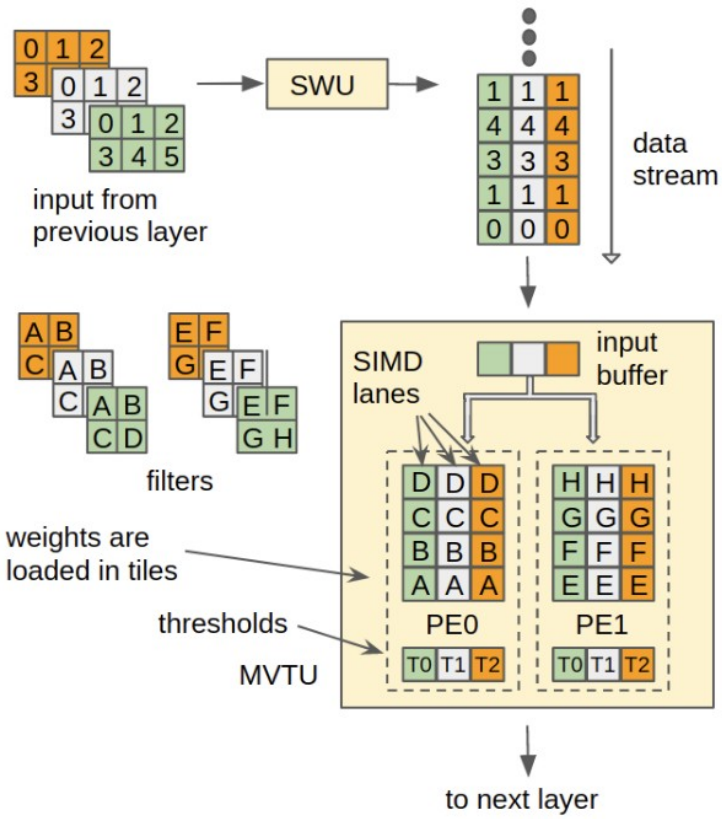


Figure 2.19: Schematic overview of the FINN architecture found in S2N2 [13]. The SWU (Sliding Window Unit) flattens the input data and forwards it to the MVTU (Matrix Vector Threshold Unit). Each Processing Element (PE) inside the MVTU processes one output channel and has a number of SIMD (Single Instruction on Multiple Data) lanes that read from input channels and multiply the input by kernel weights in parallel.

total logic occupation might be counterbalanced by those two modules. Despite efforts to find other comparisons between formal and spiking domains in terms of FPGA logic occupation, none was found. Consequently, there is a need for further work to confirm those results, and generalize to other datasets and more complex neural network topologies.

2.4.3.2 Energy comparison

On the other hand, the comparisons of SNN and FNN FPGA accelerators regarding computing needs and energy consumption have been studied on several occasions in the literature. More than comparing implementations, those works focus on building a theoretical model to evaluate the potential energy consumption reduction (or increase) when using spiking domain compared to formal domain.

In [89], Han *et. al.* proposed to evaluate energy benefits of SNNs when compared to FNNs. To do so, they proposed that energy benefits were higher at lower spike firing rates, *i.e.* a lower number of spikes means lower number of operations and hence lower energy consumption. The authors proposed to evaluate the influence of the time-window length on spike firing rate. In

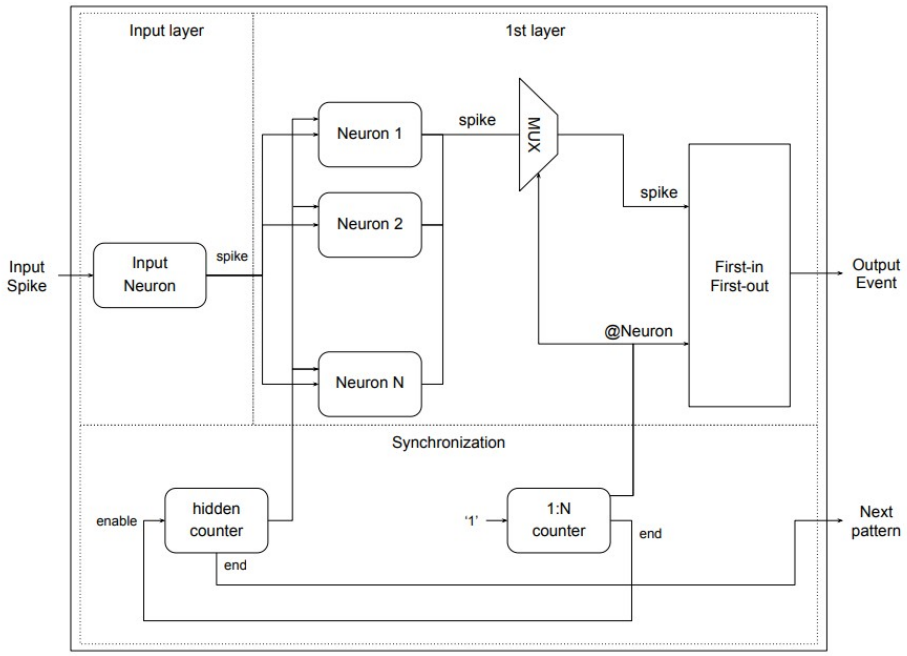


Figure 2.20: Illustration of the Neural Processing Unit of SPLEAT [14]. This NPU is used to emulate a convolution, pooling or fully-connected spiking layer in order to build a full SNN on FPGA.

timing-aware systems such as this one, time is represented as discrete increment named time-steps. Thus, the "time window" is the number of time-steps available for spike encoding. The study concluded that spike density drastically increased with respect to the number of time-steps. Thus, lower number of time-steps should be preferred in order to minimize energy consumption.

In [31], Sengupta *et. al.* proposed a novel ANN-to-SNN conversion technique for deep neural networks, such as VGG-16 [28] and deep Residual topologies. This conversion process uses threshold and weight balancing techniques in order to optimize the spiking activity. Using this method, the authors performed the first ever SNN test on the full ImageNet 2012 dataset [90], achieving 30.04% top-1 error rate and 10.99% top-5 error rate. In this paper, the authors also proposed to compare the original FNN and converted SNN regarding number of operations. The authors found that, with VGG-16 architecture, there were 1.9× more ACC operations in the SNN than MAC operations in the FNN, and 2.4× for the Residual architecture. The authors proposed this metric in order to evaluate both the relevance of their conversion technique in terms of computation reduction and energy efficiency.

In [77], the authors proposed to evaluate and compare the energy cost of formal and spiking domain in a theoretical approach. The authors focused on rate-coded SNNs, as they claim it is the most used and straightforward way to deploy SNN using conversion training technique. They proposed two equivalent hardware implementations for spiking and a formal neurons and deduced a theoretical energy cost for each of them, by dividing the processing into operations of known energy cost (multiplication, accumulation, RAM access...). The authors estimated that the spiking neuron energy consumption was lower to that of the formal neuron if and only if the number of spike per synapse was inferior to 1.72. According to the authors, such value is hardly realistic

in the case of rate-coded SNNs. They concluded that efforts must be taken to benefit from the event-based nature of SNNs. For example, they proposed to address the spike density problem by developing novel, high-sparsity spike encoding methods.

In [78], the authors proposed a compression technique associated to FNN-to-SNN conversion, aiming in minimizing spiking activity. In doing so, the authors hope to reduce energy consumption of the subsequent SNN deployment. The iterative compression technique is attention-guided: attention-maps derived from an uncompressed "teacher" model are used to prune the synaptic connections of a "student" model. The model is then converted toward spike domain, obtaining up to 33% compression rate on VGG-16 without dramatic performance loss. The authors used an IF neuron model. They also proposed a spiking back-propagation learning technique based on surrogate gradient (see 2.3.3). This method is used to re-train the model after compression and conversion, in order to reduce the number of time-steps used for inference. Using this full compression, conversion and re-training framework, the authors demonstrated near state-of-the-art classification accuracies on CIFAR-10 (91.28%) and CIFAR-100 (64.98%), using deep VGG-16 or ResNet topologies, with drastically lower spiking activity than other methods. The authors measured that their compressed SNN used 38.7 times fewer energy than an equivalent uncompressed ANN, and 12.2 times fewer energy than an equivalent ANN with identical compression on CIFAR-10 dataset. Compared to un uncompressed SNN, energy consumption was divided by 2.5. The authors also proposed a method to link energy consumption to spiking activity. Indeed, the authors approximate the energy consumption of an network to the energy consumption of its main operation: ACC for an SNN and MAC for an FNN. Counting the number of spikes, the authors were able to estimate the relative energy consumption of an FNN and a derived converted SNN. This approximation was used to give estimations of relative energy consumption in SNNs and FNNs on CIFAR-10, CIFAR-100 and Tiny ImageNet datasets. In all cases, the authors estimated that SNNs brought energy savings, and even-more when using the proposed compression and re-training technique with high sparsity targets.

2.5 Conclusion

In this section, we have presented previous works studying comparisons of spiking and formal hardware neural network implementations. In terms of logic resources usage, a single paper [88] was found, which found interesting area and power usage savings when shifting from formal to spiking paradigm. However, the literature clearly lacks sufficient comparisons: the only paper which proposes such a study focuses on a single MLP architecture, which is not sufficient to draw general conclusions. To do so, extensive comparisons covering various layer sizes, number of layers, and type of layers (convolution and pooling) must be undertaken. Moreover, this study does not take spike encoding in account, which could counter-balance the results. Finally, this comparison only covers fully-parallel implementations, and should be extended to other architectural choices and level of parallelism.

On the other hand, many papers tackle the energy-efficiency comparison between formal and spiking hardware neural networks in a theoretical manner. In all the mentioned studies, authors have drawn a strong dependence of energy consumption to spike firing rate. Accordingly, a lower number of spikes during processing means a lower energy consumption. In some of those papers [31] [78], researchers proposed to sum-up the energy consumption of an hardware NN by the number of synaptic operations. Those synaptic operations are simplified in their energy estimation models, so a formal synaptic operation is a MAC, and a spiking one is an ACC. Morevoer, au-

thors in [89], [31] and [78] proposed techniques to reduce the spiking activity at software level, by using novel conversion or training techniques. However, there is still a lack for fair and extensive comparison between hardware FNN and SNNs, covering various topologies, datasets, and architectural choices. Moreover, the comparisons are often theoretical and their is a lack of quantified comparison between formal and spiking accelerators. Additionally, the correlation between spiking activity and energy consumption must be studied further and quantified, in order to ensure the validity of such an high-level energy estimation model. Indeed, this model is much simpler than reality: it does not take memory access, level of parallelism, or other architectural choices in account.

2.6 Contributions

In this section, we provide details on the positioning and contribution of this thesis regarding the above described state of the art in Neuromorphic Engineering. Each of the following subsection is dedicated to a particular aspect of those contributions.

2.6.1 Synaptic Activity

In the literature, spiking activity is considered as a reliable metric to evaluate energy consumption in Neuromorphic systems [31] [77] [78]. It is a common idea that the energy consumption of an hardware neural network can be approximated by the cost of synaptic operations: Multiplication-Accumulation (MAC) for formal neurons and Accumulation (ACC) for spiking ones. Since the MAC operation is more energy-intensive than the other, SNNs are expected to consume less energy. However, the number of operations in an SNN is unpredictable: it depends on synaptic activity. If there is more than one spike per synapses, *i.e.* more ACCs in the SNN than MACs in the FNN the SNN might consume more energy overall. One of the contribution of this thesis is to investigate the correlation between synaptic activity and energy consumption of hardware SNNs (Chapter 4).

2.6.2 Quantitative comparison of formal and spiking domains

The bibliography proposed in this section outlines a lack of fair and extensive comparison between formal and spiking implementations of neural networks on FPGA. To our knowledge, a single paper [88] proposes a quantitative measure of the logic resources savings when using spike coding compared to formal coding. Regarding energy consumption comparison, some papers [31] [77] [78] address the issue at a theoretical level, based on the number of operations in each coding domain (ACC and MAC). However, those papers does not perform clear power or energy measure, and only produce theoretical high-level indications. Consequently, one of the objectives of this thesis is to perform an extensive comparison between formal and spiking implementations on FPGA, using various level of parallelism, and on an extensive application benchmark. Rather than directly proposing a benchmark of applications and accelerators, we propose an estimation framework. The framework provides resource, power, inference time and energy estimations for formal and spiking neural networks under two extreme level of parallelism: fully-parallel and fully-multiplexed implementations. In doing so, the framework eases the extensive comparison between formal and spiking coding domains. This framework is described in Chapter 5, and applied to a range of representative datasets.

2.6.3 Cartography of applications and neural coding domains

Some of the aforementioned datasets are images, and others are vector datasets (Fourrier Transform of voice recordings, sonar echoes...). Some have high number of classes, and some are binary tasks. This approach aims in proposing a comparison in a wide, representative range of applications. In doing so, we also investigate the conditions that seems suitable to neuromorphic acceleration. Indeed, some tasks may be suited to spikes while some others might not. The goal of this thesis is therefore to propose a cartography of neural coding domain with respect to applications. More generally, we are interested in finding the suitable conditions for spiking neural networks. Such conditions ranges from the resolution of input data to the depth of the network and the number of classes. Moreover, this cartography is complemented by the influence of parallelism on neuromorphic accelerators.

2.6.4 How to benefit from spiking domain ?

The proposed study led to several assessment regarding the energy consumption of rate-coded SNN accelerators on FPGA. In addition to the cartography of applications, we also propose several contributions and insights regarding hardware implementation of SNNs. Those contributions are:

- Neural coding domain hybridization (Chapter 6): tailoring coding domain to synaptic activity at layer level, in order to draw the best from both worlds.
- A prototype of hardware implementation of timestep-constrained SNNs trained through Surrogate Gradient Learning and using Send-on-Delta spike encoding. This method is expected to reduce synaptic activity and inference time in SNNs while maintaining state-of-the-art classification accuracy [73] [32] [91].

The development of those architectures was led by the insights provided by the study of energy consumption in neuromorphic accelerators. In both cases, the goal is to reduce synaptic activity to minimize both the number of operations (*i.e.* spikes) and temporal sparsity of spikes.

Chapter 3

Spiking Neural Networks parallel implementation: PADS

Chapter contents

In this chapter, we are going to introduce PADS (Parallel Architecture for Dense Spiking layers), an FPGA accelerator for spiking fully-connected layers. In a first section, we explain the hardware implementation of PADS in details. Then, the architecture is compared to an equivalent highly-parallel accelerator for Formal Neural Networks. This comparison brings our first insights on the cartography of suitable application domains and contexts for PADS accelerator. Moreover, a System-on-Chip (SoC) architecture is developed for PADS testing and deployment in FPGA.

3.1 Hardware Architecture

PADS is developed as a prototype for rate-based Spiking Neural Networks acceleration on FPGA. In doing so, the goal is to provide an experimental prototype for the thesis, serving for experiments and further improvements. Two major choices have been made during development: first, PADS only covers Fully-Connected layer. That is, in order to reduce development time before obtaining a prototype. Second, PADS uses a highly parallel implementation. This choice is motivated by two aspects: On the one hand, to reduce execution time and mitigate dynamic encoding drawback. On the other hand to facilitate development for the aforementioned reasons.

PADS uses the Integrate & Fire neuron model described in Section 2.3.1. The architecture comes in the form of a parameterizable VHDL code. The topology is defined in a package file, and is automatically mapped in the FPGA during synthesis. Therefore, each PADS implementation is specific to a defined MLP topology, in contrast with a programmable core such as Xilinx DPU [92]. PADS is designed to run with a clock period of 10ns (100MHz). In PADS, each neuron is physically implemented by a dedicated Neural Processing Unit (NPU). Each synapse is also physically instantiated in the design by a dedicated wire. Apart from spiking fully-connected layers of IF neurons, the architecture features a Spike Generation Cell (GenCell) for spike encoding (Section 2.3.2), and a Terminate Delta Module (TDM) for output decoding (Section 2.3.4). An overview of the architecture is presented in Figure 3.1. On this figure, each NPU is represented with a different color. The output of each NPU is connected to all downstream neurons in a fully-parallel fashion. Only two layers are represented in the figure but there can be any number of layers.

In Figure 3.1, the colored arrows are hardware synapses: 1-bit signals carrying NPU output spikes. The black arrows emerging from the Spike Generation Cell are also hardware synapses but contain an 8-bit address signal in addition to the 1-bit spike signal. The reason behind this address signal will be explained further in 3.1.1. The *input_pixel* signal carries input data in a streaming fashion, one pixel at a time in a given order. The *out_class* signal carries the identifier of the winning class, and the *stop_network* signals is triggered when the Terminate Delta condition is reached.

3.1.1 Spike Generation Cell

The Spike Generation Cell, also called GenCell, is in charge of the spike encoding of input data into spike trains following the rate-coding policy (Section 2.3.2). In the model, each input pixel is associated to an input neuron which emits spikes at a rate derived from the rate-coding equation (Equation 2.5). The GenCell emulates all input neurons in a sequential fashion. The GenCell receives a stream of input pixels (8-bit integers) at a rate of one pixel per clock cycle. The GenCell's output is made of two signals: a 8-bit address signal, and a 1-bit spike signal. The

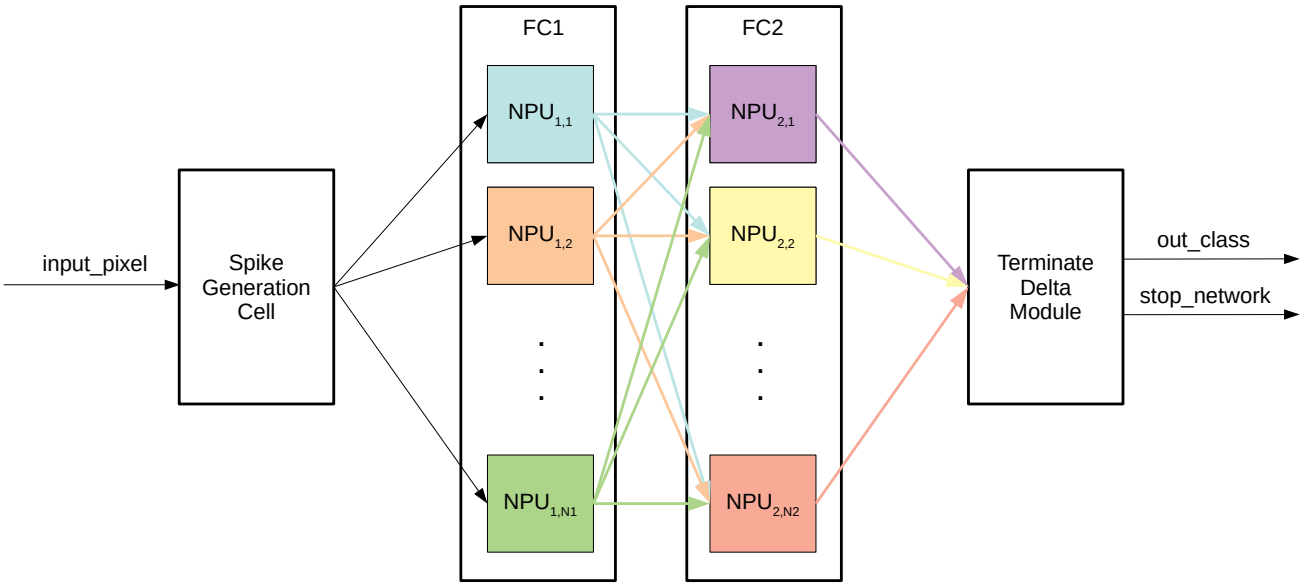


Figure 3.1: High level representation of PADS architecture, for a 2-layers spiking MLP. FCi stands for the i^{th} fully-connected layer with Ni neurons.

corresponding architecture is given in Figure 3.2, and the spike generation process is illustrated by the flowchart given in Figure 3.3. Two processing phases can be distinguished:

- The initialization phase: In the first time-step (v_time=0) the GenCell receives the input input frame in a streaming fashion, one pixel per clock cycle. Each pixel is associated with a period corresponding to the frequency of the spike train to be generated (Equation 2.5).
- The spike emission phase: On the following time-steps (v_time¿0), the input periods are compared to the current v_time. A spike is emitted when $v\_time \equiv 0$ modulo P (P the spike train Period). In other words, a spike is generated when the remainder of the division between v_time and P is zero.

Before going in more details, it should be noted that the rate-coding policy is pre-computed off-line to simplify the implementation. Pixels being 8-bit encoded, there are 256 possible pixel values and thus 256 possible spike train periods. The 256 periods are computed off-line and stored in a hard-coded LUT. The period corresponding to a pixel value i is stored at the i^{th} cell of the aforementioned LUT.

3.1.1.1 Double Counter

The GenCell features a nested double counter which paces the spike generation process. The outer counter controls discrete "virtual" time increments. Indeed, the spike train periods are defined in terms of "virtual" time-steps rather than real time. This "virtual" time is referred to as v_time in the following. At each v_time increment, an inner counter C_{pixel} scans the input neurons, at a rate of one per clock cycle and generates a spike if $v\_time \% P = 0$. For example, the GenCell time-step length (*i.e.* v_time increment) on MNIST is 784 clock cycles (28x28p samples). The value of C_{pixel} thus points to the current active input neuron (*i.e.* input pixel). Moreover, the output address signal (o_addr) always carries the value of C_{pixel}. In all, the double counter acts as a finite state machine which controls the GenCell behavior.

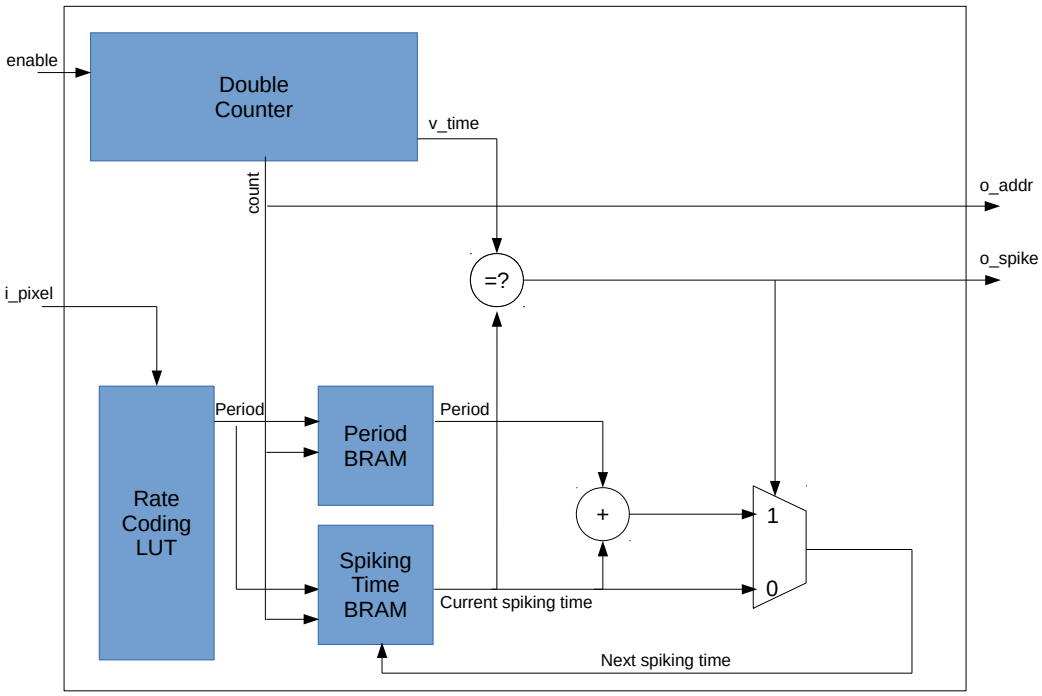


Figure 3.2: Block Diagram of the Spike Generation Cell.

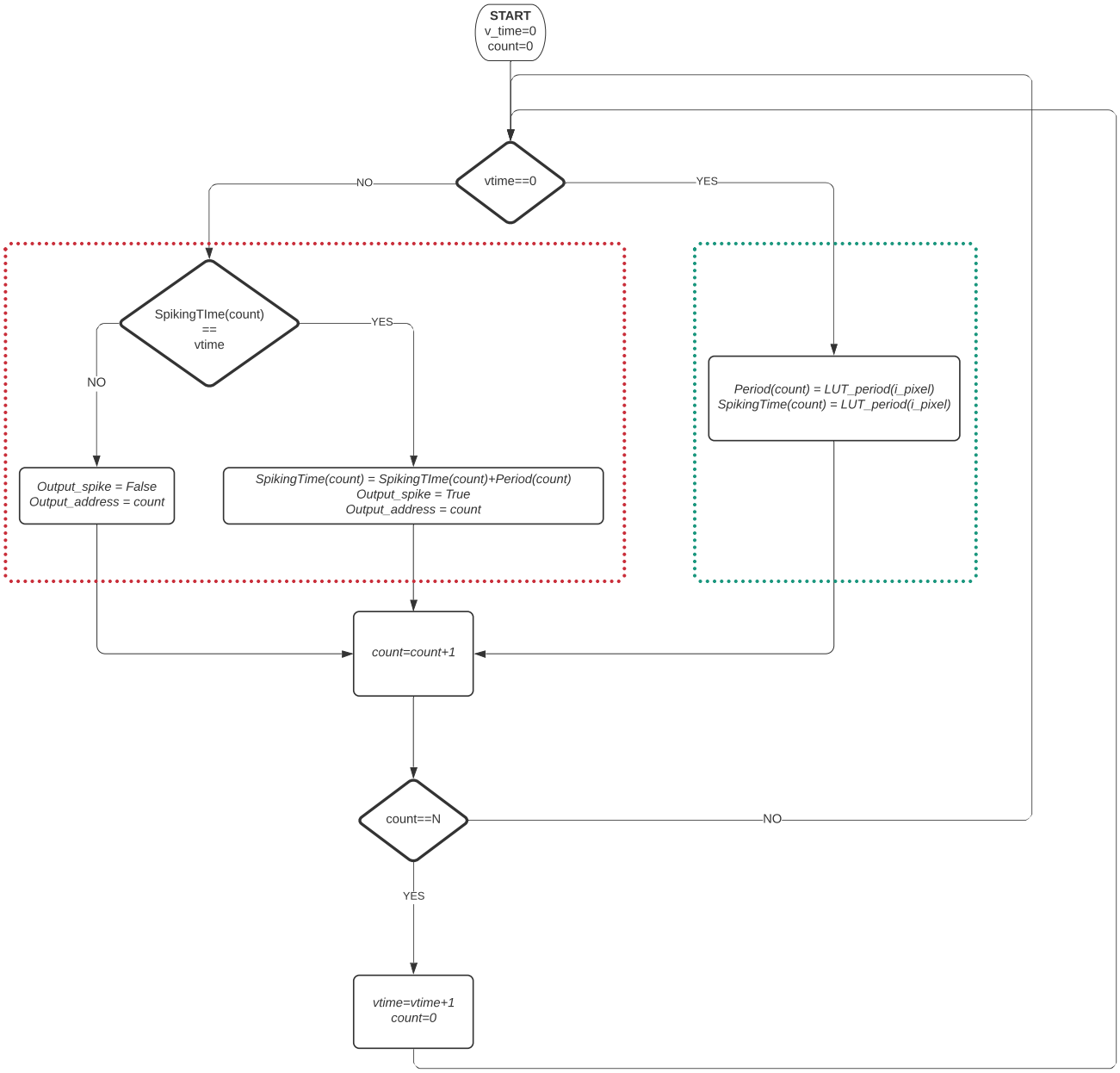


Figure 3.3: Spike emission process flowchart. Initialization is highlighted in green, emission in red.

3.1.1.2 Initialization Phase

The initialization phase occurs when *v_time*=0. During this phase the GenCell receives the input data stream at a rate of one pixel per clock cycle. The stream must be synchronized with the C_{pixel} counter so that each input neuron can be associated to the right pixel value. For each pixel, the corresponding spike train period is retrieved in the rate-coding LUT. The subsequent period value is stored in the Period LUT (P-LUT). Additionally, a copy of the value is stored in the Next Spiking Time LUT (NST-LUT) as explained in the following. The address in the LUT is given by the neuron counter (C_{pixel}). The NST-LUT is used to prevent the use of a hardware-intensive modulo operator. That is, the NST-LUT directly indicates the "next-spiking time". When C_{pixel} reaches the last pixel, *v_time* is incremented and the Spike Emission Phase begins.

3.1.1.3 Spike Emission Phase

The spike emission phase begins when *v_time*>0. The process is written in pseudo-code in Algorithm 1, and is also shown in the red section of the flowchart in Figure 3.3.

Algorithm 1
Pseudo-code of the Spike Emission Phase in the GenCell process

```
v_time ← 1
while stop_network ≠ True do
    C_pixel ← 0
    for n in [[0, N_pixel−1]] do
        if NST_n = v_time then
            o_spike ← True
            o_addr ← C_pixel
            NST_n ← NST_n + P_n
        else
            o_spike ← False
        C_pixel ← C_pixel + 1
    v_time ← v_time + 1
```

At each *v_time* increment, the GenCell scans all the input neurons one by one. For a given neuron n ($C_{pixel} = n$), the Next Spiking Time NST_n is retrieved from the NST-LUT. The period P_n is retrieved from the P-LUT. If *v_time*= NST_n, a spike is emitted. It should be noted that the corresponding neuron address ($C_{pixel} = n$) is carried by *o_addr* output signal. The Next Spiking Time is incremented by P_n and the new value is stored in the n^{th} cell of the NST-LUT. This whole process has been developed to avoid using a modulo operator, which is not adapted to hardware implementations and caused timing violations at 100MHz. This process is repeated for each input neuron. When the last neuron is reached, *v_time* is incremented and the process starts again from the first neuron until reaching the termination condition explained in Section 3.1.3.

3.1.2 Neural Processing Unit

The Neural Processing Unit (NPU) is in charge of emulating a single IF spiking neuron. There are two types of NPUs: one is specifically designed for the first layer, and one is generic for all the

other layers. In our model the membrane potential accumulator is not reset after a spike emission. The IF threshold θ_{IF} is rather subtracted from the accumulator.

3.1.2.1 Generic NPU

The generic NPU (G-NPU) architecture is made of three stages: the weight-affectation stage, the integration stage, and the accumulator & threshold stage. This architecture is represented in Figure 3.4 for an example neuron with 4 input synapses.

The weight affectation stage is composed of one multiplexer per input synapse: if a synapse is active (*i.e.* a spike is received though that synapse), the multiplexer forwards the corresponding synaptic weight and 0 otherwise. It should be noted that the synaptic weights are hard-coded in the G-NPU design. Then, the resulting set of weights and zeros are summed together in the integration stage. Summation is performed by a 2-by-2 pipelined adder-tree. In doing so, the architecture is able to process 1 spike per input synapse and per clock cycle. Register barriers after each stage ensure synchronization and limit the critical path of the design. The adder-tree is automatically generated thanks to hand-coded VHDL functions and packages.

The resulting sum is added to the membrane potential accumulator (orange register in Figure 3.4). If the result is above the neuron threshold (Θ_{IF}), a spike is emitted and Θ_{IF} is subtracted from the membrane potential accumulator. Otherwise, no spike is emitted and the membrane potential remains unchanged. This process is pipelined in the same fashion than the rest of the architecture. Its implementation is described in Algorithm 2.

Algorithm 2
Pseudo-code of the PADS NPU accumulation & threshold stage process.

$potential\_spike(\text{t}) \leftarrow potential(\text{t-1}) + val(\text{t}) - \Theta_{\text{IF}}$
$potential\_nospike(\text{t}) \leftarrow potential(\text{t-1}) + val(\text{t})$
if $potential(\text{t-1}) > \Theta_{\text{IF}}$ **then**
    $o\_spike \leftarrow$ True
    $potential(\text{t}) \leftarrow potential\_spike(\text{t-1})$
else
    $o\_spike \leftarrow$ False
    $potential(\text{t}) \leftarrow potential\_nospike(\text{t-1})$

In the pseudo-code, $Val(\text{t})$ is the adder-tree output at clock-cycle t, and $potential(\text{t})$ is the membrane potential at clock-cycle t. $potential\_spike(\text{t})$ is represented by the red register on Figure 3.4, and $potential\_nospike(\text{t})$ by the orange one.

3.1.2.2 First Layer NPU

The first layer NPU (FL-NPU) is based on the generic architecture presented above, but optimized for the first layer. As seen in 3.1.1, the GenCell is sequential and emits at most one spike per cycle. The Gencell also features an address signal which carries the address of the emitting neuron. Thus, the input parallelism featured in the generic NPU is useless in this context: only one synapse will be stimulated at a time. A simplified version is therefore designed to save hardware resources. As this layer is often the biggest in an MLP topology (highest number of neurons and synapses), this optimization might bring interesting resource savings (Section 3.2). The FL-NPU architecture is given in Figure 3.5. This architecture is made of two stages: the weight affectation stage and

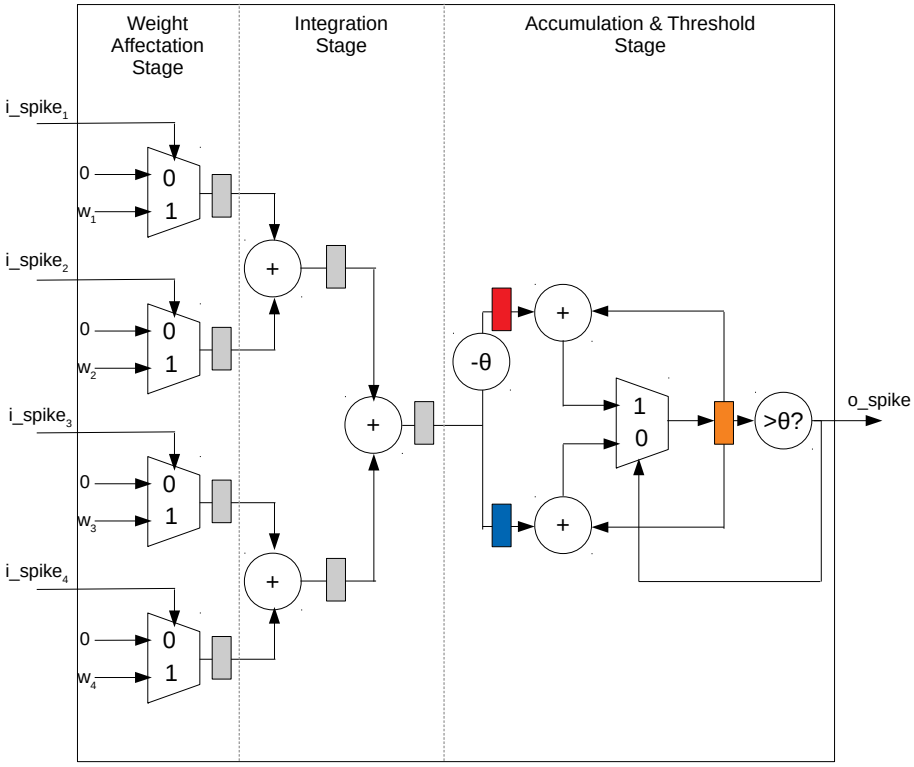


Figure 3.4: Architecture of a generic NPU in PADS, represented with 4 input synapses. Register barriers are represented as rectangles.

the accumulation & threshold stage. In the weight affectation stage, the weight is retrieved in a Look-Up-Table (LUT) using the address signal (i_addr). It should be noted that the weight LUTs are stored in Block RAM during synthesis, in contrast with the G-NPU in which the synaptic weights are hard-coded. The accumulation & threshold stage works exactly like in the generic NPU design.

3.1.3 Terminate Delta Module

The Terminate Delta Module (TDM) is in charge of the Terminate Delta procedure introduced in Section 2.3.4. The TDM counts output spikes on the last layer of the SNN, and determines when a neuron has spiked Δ times more than any other neuron. Δ is a user-defined parameter. In some cases, the Terminate Delta condition is never achieved, and the network runs indefinitely. This problem arises when there are not enough output spikes, or when the concurrence between several neurons is too fierce. To cope with this issue, the TDM also features a timeout mechanism which forces the network to stop after a certain number of clock cycles. The TDM implementation is pipelined like all the other modules, and is able to receive one spike per synapse at each clock cycle. The architecture is illustrated in Figure 3.6, where the timeout implementation and register barriers are not shown for the sake of clarity.

The TDM is made of 4 stages: a counter stage, two trees of max operators, and the delta stage. First, the counter stage counts the number of output spikes on-the-fly. The counts are stored in an array of registers indexed in the same order as the output neurons. In the second stage, a pipelined tree of maximum operators is used to find the index of the maximum value in the array. This value (Max_1) and its index (ID_Max_1) are stored in registers. Max_1 is replaced by

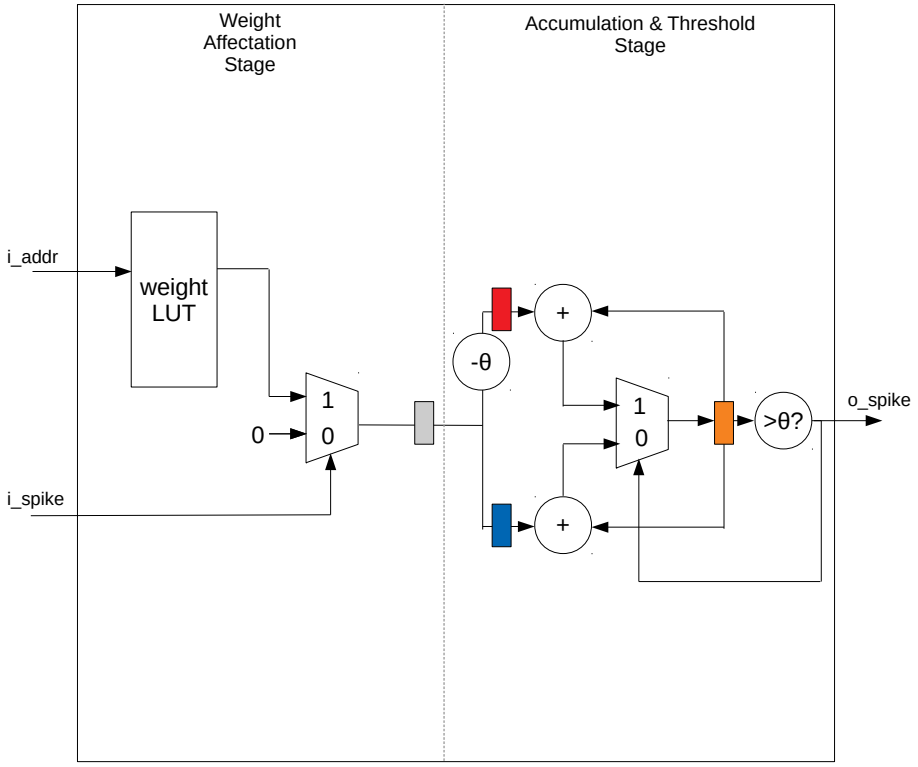


Figure 3.5: Architecture of a PADS Input-NPU, optimized for the input FC layer

a zero in the array, which is passed to the next maximum-tree stage. This second pipelined tree of maximum operators finds the maximum value (Max_2) in the new array. At this phase, the TDM has thus retrieved the maximum spike count (Max_1), the second maximum spike count (Max_2), and the index of the most spiking neuron (ID_Max_1). Those three values are passed to the delta stage. The absolute value of Max_1-Max_2 is computed. If it is greater than Δ, the Terminate Delta condition is reached and ID_Max_1 is the winning class. Otherwise, the process continues until the condition is met or the timeout procedure forces the network to stop.

3.2 Hardware Synthesis Results

In this section, PADS is synthesized on FPGA. We analyze the hardware synthesis results in terms of logic resources usage, power usage, execution time and energy consumption. PADS will be compared to the VGT architecture, a parallel accelerator for Formal Neural Networks described in Section 2.2.2.1. The VGT architecture uses the same level of parallelism and implementation choices than PADS. Like in PADS, each neuron of the model is hard-coded in the design. VGT neurons have a similar structure made of three stages: A weight affectation stage (MAC operation), a tree of adders for input activity integration, and a ReLu stage for the activation function. The whole neuron is pipelined like PADS. This similarity enables a fair comparison. Some additional experiments on PADS are also presented in a second time. Those experiments aim at understanding the influence of various SNN-specific hyper-parameters on SNN acceleration. In the remaining of this document, we oppose static and dynamic processing. Dynamic refers to time-based computation, like in rate-based SNNs. On the other hand, FNNs are static models.

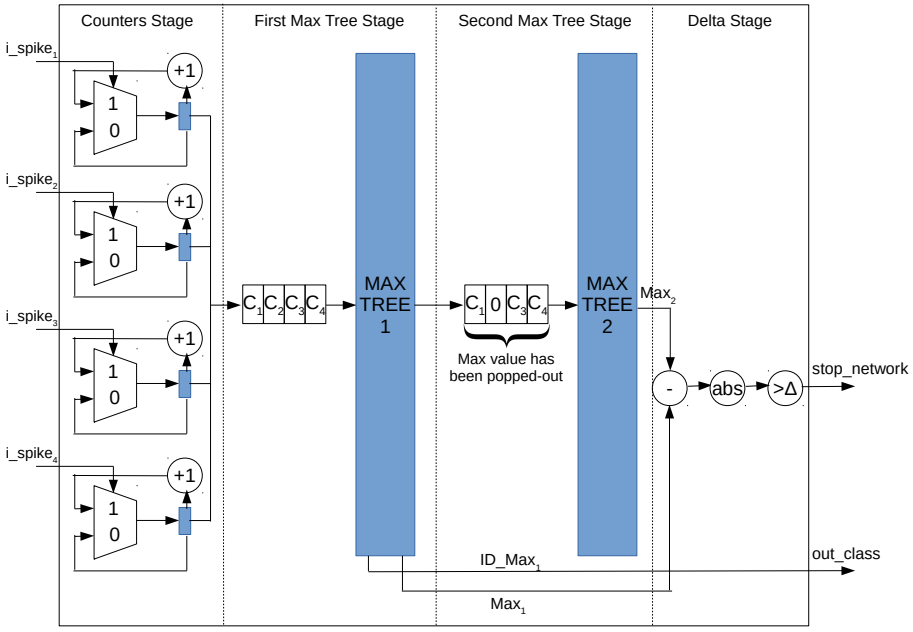


Figure 3.6: Architecture of the PADS' Terminate Delta Module with 4 output neurons.

3.2.1 Methodology

PADS and VGT architectures are applied to the OPS-SAT [93] classification task. A simple MLP topology is used. The Terminate Delta Module is configured with $\Delta = 4$. The GenCell is configured with $MaxPeriod = 10$ and $MinPeriod = 1$.

3.2.1.1 Dataset Specifications

The grayscale OPS-SAT dataset is made of 28x28p grayscale patches extracted from 1920x1080p RGB satellites images taken by the OPS-SAT embedded camera (See Chapter 6.3.1.1 and Appendix 8.2.2). The original satellite images have variable light expositions, noise levels and orientations. The task is to classify each patch as "*cloud*" or "*no-cloud*". The dataset is made of 100 860 patches extracted from 16 full-size HD pictures. There are 40 078 *cloud* patches and 60 782 *no-cloud* patches. The dataset is split in two random batches, with 90% for training and 10% for testing.

3.2.1.2 Training with TensorFlow Keras

A Multi-Layer Perceptron (MLP) with topology 28 × 28 - 100 - 2 is used for classification. The MLP is trained on this dataset using TensorFlow with Keras front-end. The training hyper-parameters are listed in table 3.1. The synaptic weights are clipped between −1 and 1. This weight clipping is used before conversion towards spiking domain in order to balance weights and neuron threshold (Θ_{IF}). This method is inspired from N2D2 [1], and is analogous to a weight-balancing technique [82]. In addition, a Softmax [94] activation function is used in the output layer during training in formal domain. The test score obtained in formal domain using TensorFlow is 71%. For test in spiking domain, the TF-trained weights are exported to N2D2 for spiking inference. The result spiking test score is also 71% with $\Delta = 4$. For comparison, the original formal CNN obtained 81% accuracy on RGB patches (instead than grayscale).

Table 3.1: TensorFlow Keras parameters used for training in all our experiments.

Parameter	Value
Loss	Categorical Crossentropy
Optimizer	Adam
# Epochs	50
Batch Size	128
Validation Splot	0.1

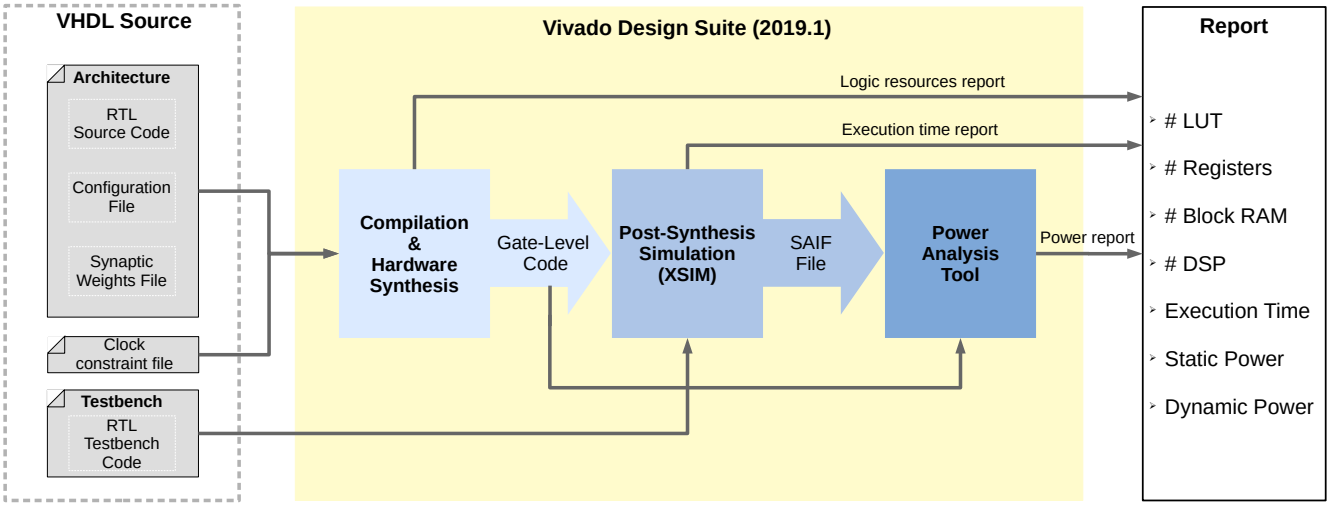


Figure 3.7: Workflow for synthesis, simulation and reporting using Vivado Design Suite (2019.1). The source code (VHDL) is depicted on the left, and the resulting report on the right.

It should be noted that N2D2 has been widely used for SNN experiments during this Phd thesis as it was one of the only framework to offer both training and SNN conversion at the time work began. Moreover, N2D2 is developed by the CEA LIST which is part of the french technological research ecosystem.

3.2.1.3 Hardware Synthesis & Power Estimation

After learning, the trained synaptic weights are exported as VHDL packages for PADS and VGT architectures. PADS and VGT are both configured with the same 28×28 - 100 - 2 MLP topology. Both architectures are synthesized separately using Vivado Design Suite, the Xilinx framework for hardware synthesis, targeting the XCZU9EG family device of a ZCU102 board. The synthesis, simulation and reporting workflow is illustrated in Figure 3.7. A post-synthesis simulation is performed afterwards using the Vivado XSIM simulation tool. Vivado XSIM tool uses a custom VHDL testbench to provide input signals for the design. A specific testbench is developed for each architecture. For PADS the data is streamed by the testbench as required by the GenCell design. For VGT on the other hand, the pixels are injected all at once as specified in the VGT design. The aim of this realistic simulation is to generate a SAIF (Switching Activity Interchange Format) file. This file contains log of all signal switches occurring in the architecture during simulation. It is later used in the Vivado Power Analysis tool in order to estimate the power usage of the IP

Table 3.2: Hardware resources measures for PADS and VGT MLPs on OPS-SAT Grayscale Dataset: 28×28 - 100 - 2

Module	**GENCELL**	**FC1**		**FC2**		**T_DELTA**	**TOTAL**	
Architecture	**PADS**	**PADS**	**VGT**	**PADS**	**VGT**	**PADS**	**PADS**	**VGT**
LUT (#)	9 944	7 662	940 659	1 574	1 776	138	19 318	942 435
FF (#)	283	2 798	671 168	2 065	1 772	85	5 231	672 940
BRAM (#)	1	0	0	0	0	0	1	0
DSP (#)	0	0	755	0	40	0	0	795
DYNAMIC POWER (mW)	196	39	4 465	12	12	1	248	4 477

based on its simulated internal activity. The synthesis results for both VGT and PADS are given in Table 3.2.

3.2.2 Comparison with VGT

At first glance, Table 3.2 clearly shows that logic resources requirements are lower for PADS.

3.2.2.1 DSP usage

First, VGT uses 795 DSPs, whereas PADS doesn't use any. This substantial gap is directly related to the fundamental difference between formal and spiking coding domain. As mentioned in Section 2.3.1, SNNs integrate information via simple ACC operations, whereas FNNs require MAC operations. During synthesis, Vivado maps each parallel MAC operation on a DSP (Digital Signal Processor). DSPs are specifically optimized for that purpose. On the other hands, ACC operations are directly mapped in the Programmable Logic. DSP being a scarce resource in FPGAs, PADS seems preferable in that regard.

3.2.2.2 LUTs and Registers in FC layers

Second, VGT uses 49 times more Look-Up-Tables (LUTs) and 129 times more registers (also called Flip-Flops, FF). As shown in Table 3.2, this significant difference is mostly due to the first layer (FC1). For FC1 specifically, VGT uses 123 times more LUTs and 240 times more FFs than PADS. On the other hand the resource usage of the second layer (FC2) is similar in PADS and VGT. Indeed, there are two types of Neural Processing Units (NPU) in PADS: a FL-NPU optimized for the first layer, and a G-NPU for all the others. As explained in 3.1.2 the G-NPU has a large input parallelism which implies an adder-tree for input integration. Instead, the first layer NPU only has a single input and an addressing mechanism which only requires one adder. Consequently, the G-NPU is much more resource intensive.

3.2.2.3 Power Usage

Third, VGT uses 18 times more power than PADS overall, and the overhead is mostly due to FC1, where VGT uses 114 times more power than PADS. This observation matches with the resource usage results. This is quite straightforward, as more resources implies more signals and thus, more signal toggles during execution. This in turns implies a higher power usage. Moreover, computation is sparse in PADS due to the rate-coding policy. This implies a low signal toggle rate

in the design. On the other hand, VGT works on static data, thus the signal toggle rate is much higher. This difference explains the power usage difference.

So, PADS provides interesting resource and power savings compared to VGT. The absence of DSPs in PADS design provide a better scalability, since this resource is scarce on FPGA circuits. The lower power usage could also benefit to low-power applications, such as in solar-powered systems.

3.2.3 Spike Generation Overhead

Table 3.2 highlights the resource overhead induced in PADS by the GenCell and the Terminate Delta Module (TDM). Indeed VGT does not requires such specific modules, dedicated to spike encoding and decoding. Concerning the TDM, the overhead is not very significant: it is only accountable for 0.7% of LUTs and 1.6% of registers in PADS. On the other hand, the overhead caused by the GenCell is much noteworthy. It represents 51.4% of LUTs, and 6.5% of registers of the total design (GenCell+PADS+TDM). Thus, optimizing the GenCell is interesting to further reduce the hardware footprint of PADS.

3.2.3.1 Execution Time & Energy Overhead

The resource overhead is not the only issue with this module. The sequential output allows using a hardware-friendly FL-NPU in the first layer. However, the drawback on execution time is very significant. PADS takes 8000 clock cycles in average for classification of OPS-SAT 28x28 grayscale patches, whereas VGT takes only 23. At 100MHz this is equivalent to $83\mu s$ for PADS and $230ns$ for VGT. PADS energy consumption is thus 2.1×10^{-5}J per image in average, whereas it is only 6.5×10^{-7}J in VGT. In other words, PADS is 32 times more energy intensive than VGT. Thus, PADS architecture does not comply with our initial intuition that spiking domain mitigates energy consumption. This observation is entirely due to the execution time overhead. It should be noted that this is not a strict disadvantage for PADS. The energy consumption is higher, but spread across extended period of times. Thus, PADS is compliant with low-power applications, such as solar-powered systems (satellites...).

3.2.3.2 Causes of the timing Overhead

The difference in execution time is quite straightforward: VGT works in a static and synchronous fashion, where all inputs are presented at once for each layer. Oppositely, PADS works in a dynamic and asynchronous fashion because of the event-based nature of spiking domain. This mechanism intrinsically results in higher execution time, and is even more visible when using a sequential spike-encoding module like the GenCell. As mentioned earlier in Section 3.1.1, the GenCell scans the input vector entirely at each time-step, which results in timing overhead.

In the remaining of this subsection, we propose two experiments to investigate the behavior of the GenCell. More specifically, we study the influence of spatial and temporal resolution of spike encoding on the behavior of the GenCell. Temporal resolution refers to the number of periods available for spike encoding. Spatial resolution refers to the input size. The aim of those experiments is also to evaluate a potential trade-off between execution time and prediction accuracy.

3.2.3.3 Influence of Temporal Resolution

As a reminder, the rate-coding formula is given in Equation 3.1.

$$\begin{aligned} \mathrm{P} &= \frac{1}{\mathrm{MaxFreq} + (\mathrm{MinFreq} - \mathrm{MaxFreq}) \times v}, \\ v &= \frac{1 - i}{255} \end{aligned} \tag{3.1}$$

Where P is the spike train period associated to a pixel of value i (8-bit integer), MaxFreq and MinFreq are respectively the maximum and minimum spike train frequencies.

As mentioned in Section 3.1.1, the GenCell operates in a time-step based fashion rather than real-time. Spike train periods are thus expressed in terms of time-steps. According to Equation 3.1, spike trains are encoded with periods ranging from MaxPeriod $= \frac{1}{\mathrm{MinFreq}}$ to MinPeriod $= \frac{1}{\mathrm{MaxFreq}}$. The difference between MinPeriod and MaxPeriod characterizes the temporal resolution of spike encoding. There is thus a finite number of possible periods between MinPeriod and MaxPeriod. Those parameters must have a strong influence on execution time as they govern the spike emission rate. MinPeriod is set to 1, so that spike trains are able to begin at the first time-step. With a fixed MinPeriod = 1, *MaxPeriod* defines the temporal resolution by itself.

Under those conditions, Figure 3.8 shows the influence of MaxPeriod, *i.e.* temporal resolution. In this experiment PADS is configured with a small mono-layer 784-10 topology. The classifier is applied to MNIST dataset. TensorFlow Keras is used for training with the parameters listed in 3.1. The weights are exported in a VHDL package, and PADS is configured with the 784-10 topology. For each value of MaxPeriod, we perform behavioral simulation at Register Transfer Level (RTL) using Modelsim, on a hundred MNIST samples. We measure the average prediction accuracy, number of input & output spikes, and execution time with respect to MaxPeriod.

Figure 3.8 shows that high MaxPeriod values (*i.e.* high temporal resolutions) offers significantly better accuracy, lower execution times and lower number of input spikes. However, the number of output spikes is slightly lower for low temporal resolutions. Indeed, low temporal resolution implies that fewer periods are available for spike encoding. In other words, very different pixels might be encoded with identical spike trains. Thus, the network behavior is deteriorated which explains the heavily degraded accuracy (solid black line) at low MaxPeriod values, and the increased execution time. The execution time is higher because of the altered behavior: the Terminate Delta Module has more difficulty to determine the winning class under such conditions. Moreover, lower MaxPeriod value does not only imply lower resolution but higher input spike rates as shown in the Figure (red bars). As explained previously, power usage is directly related to signal toggles in the design. Higher spike rates results in higher signal toggle rates which in turns increase dynamic power usage.

Consequently, high MaxPeriod values (*i.e.* high temporal resolutions) can help mitigating execution time, and further reduce power usage. However, execution time remains very high with 9000 clock cycles at best. This execution time is even higher than with the 784-100-2 topology used for OPS-SAT grayscale patch classification (8000 clock cycles). Consequently, smaller topologies does not imply lower execution times. Fine tuning the temporal resolution does not enable to compete with VGT either, in terms of processing speed.

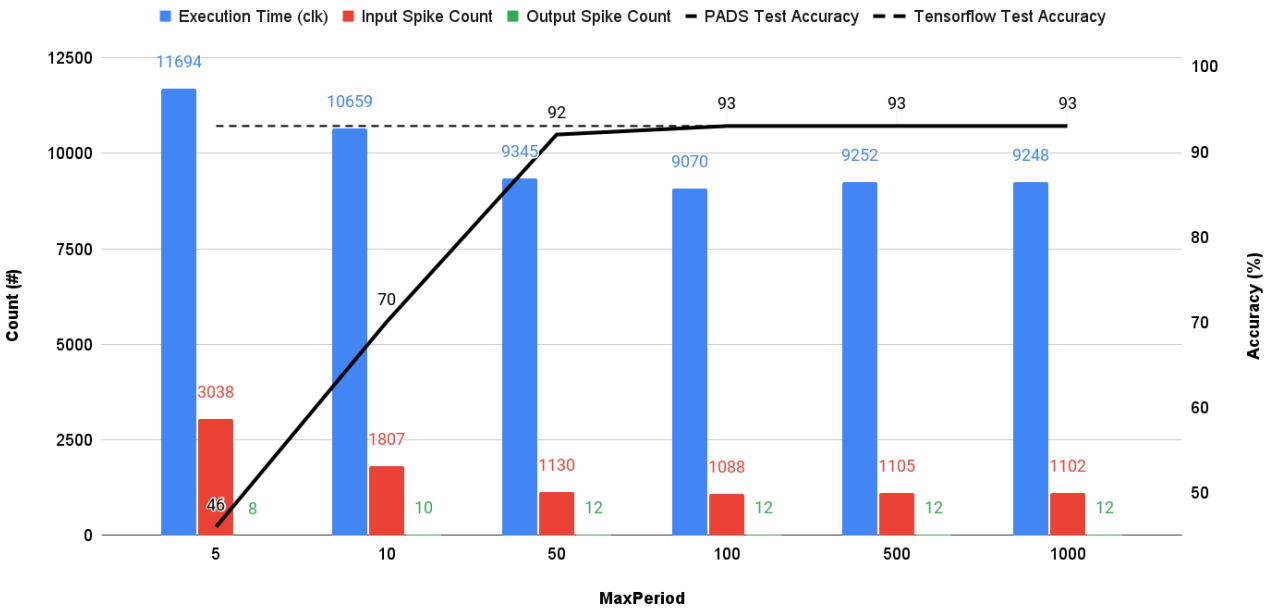


Figure 3.8: Evolution of test accuracy, number of input & output spikes and execution time (in clock cycles) with respect to MaxPeriod. All measurements are averaged on 100 samples. Topology: 784-10 on MNIST.

3.2.3.4 Influence of Spatial Resolution

In this paragraph we study the influence of spatial resolution on the GenCell behavior and subsequent execution time. Because of the sequential nature of the GenCell, the time-step length depends on the input size. The process requires to scan the whole "Next Spiking Time" vector (NST-LUT) (Section 3.1.1) at each time-step at a rate of one element per clock cycle. For example, on MNIST dataset, the time-step length is 784 clock cycles. Quite straightforwardly the execution time could be mitigated by reducing the input size. In other words, input down-sampling could provide interesting speed-ups at the cost of a loss of information.

For that purpose, we study the influence of input down-sampling on PADS behavior. A single FC layer SNN is used with I_{size} inputs and 10 outputs. I_{size} is the input size varying from 28 to 10. A specific network is trained for each input size using Tensorflow Keras with the parameters given in 3.1. A *Resize* layer from Keras front-end is used at the input. This layer down-samples the input data to the required I_{size} value. Moreover, the resized samples are also exported in VHDL testbenches for later simulation. Each set of trained weights is exported in a dedicated VHDL package and PADS is configured successively with each topology. Each iteration is simulated at RTL level using Modelsim, using the previously exported down-sampled data. Each simulation covers a hundred of such samples. We measure the variations of average accuracy, number of input & output spikes and execution time with respect to input size. The results of this experiment are shown in Figure 3.9. It should be noted that we use MinPeriod = 1 and MaxPeriod=100.

The MLP topology changes with input size, thus the network is retrained each time. Thus, variations of testing accuracy are not only caused by spatial resolution. Intrinsic variability of training also plays a role. Therefore, the difference between formal and spiking accuracy should be studied, rather than spiking test accuracy only. In doing so, the training variability is mitigated. This difference is referred to as "accuracy gap". The accuracy gap increases towards low input

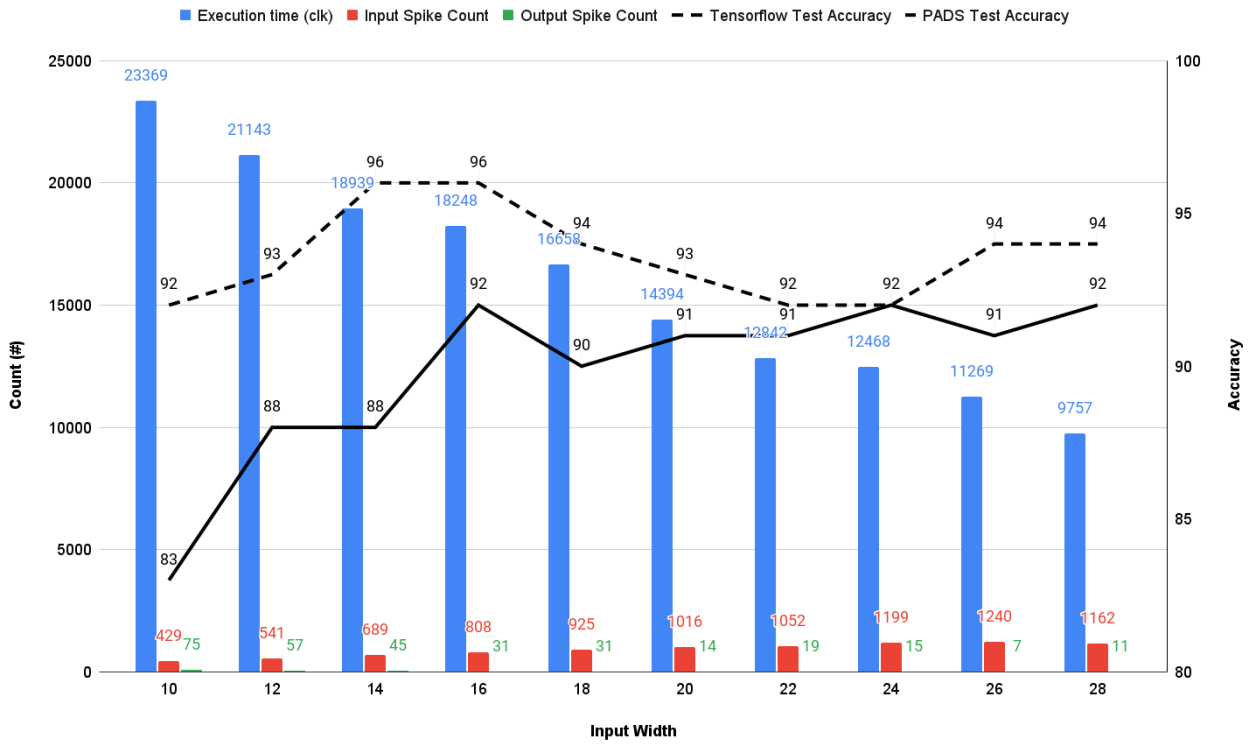


Figure 3.9: Evolution of test accuracy, number of input & output spikes and execution time (in clock cycles) with respect to input size. All measurements are averaged on 100 samples. Single FC layer SNN on MNIST dataset with various input sizes.

sizes (*i.e.* low spatial resolutions). Accordingly, the converted SNN supported by PADS seems less robust to down-sampling than the original FNN tested in TensorFlow.

Moreover, smaller input sized means fewer neurons in the input layer. Fewer neurons in turns generate fewer spikes thus the number of input spike per image (red bars) increases with respect to input size. On the other hand, the number of output spike per image is higher for low input sizes. Thus, at low spatial resolutions, fewer input spikes generates more output spikes than at high spatial resolutions. This phenomenon confirms the degradation of the network's behavior for higher down-sampling rates. As in the previous experiment (temporal resolution), loss of information also increases execution time. Consequently, reducing spatial resolution through down-sampling of input data does not mitigate latency, and heavily degrades accuracy. Thus, down-sampling is not viable to reduce latency in PADS. This experiment also shows that PADS is more affected by spatial down-sampling than the original TensorFlow model.

3.2.4 Conclusions on PADS hardware implementation

In this section, PADS is compared to VGT on OPS-SAT grayscale dataset with a $784 - 100 - 2$ MLP. Under such conditions, PADS enables drastic logic resources savings relatively to VGT. The most significant difference lies in DSP usage. If VGT uses a lot of DSPs for MAC operations, PADS uses only LUTs and Flip-Flops for ACC operations. A major difference was also found in the first layer, where the specific NPU enabled drastic resources savings in terms of LUTs and

Registers. At the same time, PADS provides substantial power savings relatively to VGT with 248mW and 4477mW respectively (18 times less).

However, the dynamic nature of information and the sequential behavior of the GenCell are responsible for the timing overhead in PADS. Indeed, PADS is in average 30 times slower than VGT on the OPS-SAT grayscale dataset. Despite power savings, it consumes in average 2.1×10^{-5}J per image, against 6.5×10^{-7}J for VGT. PADS uses more energy, but spread across a longer processing time. Fine-tuning the temporal resolution of rate encoding mitigates the GenCell latency. However, VGT still remains several dozens of times faster than PADS, even with optimal temporal resolution. On the other hand, down-sampling the input data does not mitigate latency: the loss of spatial information seems to make the classification task harder, increasing the execution time. Additionally, one simple way to solve the temporal sparsity of spike encoding could be a parallelization of the GenCell process. For this purpose, a fully-parallel version of the GenCell was developed with simplified hardware implementation. After preliminary experiments on MNIST, the parallel GenCell saturates the Zedboard resources on its own. Therefore, this approach is unrealistic. On the other hand, proposing an intermediate level of parallelism would have required too much development for the purpose of our prototype. Moreover, this feature would have been incompatible with the NPU design of PADS.

In the light of these considerations, a first cartography of favorable application domains for PADS arises. PADS is more suited to systems constrained by the resource (*e.g.* drones, satellites..) or power budget (*e.g.* solar or wind powered systems), whereas VGT is more suited to systems constrained by energy (*e.g.* battery-powered systems) or time (*e.g.* real-time applications).

Because PADS is more resource and power efficient, it is preferable to use VGT in the context of highly constrained solar-powered embedded systems, such as Satellites. Indeed, size, weight and power are the most limiting factors, rather than overall energy consumption. The logic resource and dynamic power savings implies that it might be used in more constrained systems than VGT would which are limited by instantaneous power rather than overall energy consumption. This is also true for other types of solar powered devices. On the other hand, PADS is not adapted to battery-powered systems, as energy becomes the limiting factor in such conditions. PADS seems less adapted adapted to real-time systems either because of its increased execution time. At this point it should be noted that all presented results are constrained to rate-coded MLP deployed on highly parallel hardware. Moreover, they are constrained to OPS-SAT classification task. In Chapter 5, those results are generalized to low-level of parallelism and a benchmark of 7 datasets (images and 1D signals).

3.3 Conclusion

In this Chapter, we have presented the PADS architecture. PADS is a highly-parallel Spiking MLP architecture featuring a Spike Generation Cell, two variants of Neural Processing Units, and a Terminate Delta Module. PADS was developed as a first prototype of SNN hardware accelerator in the thesis, serving as a toolbox to investigate such implementations and support later improvements. PADS was compared to an equivalent highly-parallel formal architecture: VGT. During this comparison, PADS demonstrated drastic logic resources and power savings compared to VGT. However, the dynamic nature of rate-coding and the sequential implementation of the Spike Generation Cell were found responsible for a severe increase in execution time. Consequently, PADS energy-intensiveness was a lot greater than that of VGT. However, in PADS, this energy consumption is spread across time.

This observation led us to our firsts insights concerning the cartography between applications and coding domain. PADS seems well suited to highly resource and power constrained systems, such as satellites. Indeed, such systems have drastic constraints in terms of size and weight. Saving resources is thus helpful in that regard. Moreover, solar-powered devices like satellites are limited in terms of instantaneous power usage rather than overall energy consumption. VGT does not comply with such constraints because of the high instantaneous power usage when PADS seems well suited to this type of systems. On the other hand, VGT is much more adapted for battery-powered systems, where overall energy consumption is the limiting factor. Moreover, PADS seems less adapted to real-time systems because of its long response time.

However, those insights are only true for very specific cases. Only two architectures where compared, which does not covers other implementation choices, such as sequential architectures. Moreover, this comparison only covered a single MLP topology, and a single application (OPS-SAT grayscale patch classification). Additionally, this comparison only stands for conversion-based and rate-based SNN models. In Chapter 7, we address a novel type of SNN models. Since rate-based converted SNNs are responsible for a significant inference time overhead, we have indeed looked for alternative choice in the literature. Moreover, our experiment concerning the spatial resolution influence on PADS did not provide the expected results. Indeed, reducing spatial resolution was intended to reduce execution time, but the opposite phenomenon arose. This finding was imputed to loss-of-information. Consequently, the idea of Neural Network Hybridization emerged. It will be introduced in details in Section 6.

In addition to this work, a SoC design was developed to support PADS hardware deployment. The role of this platform is to support hardware deployment of PADS and other hardware accelerator on FPGA. In doing so, it provides a useful testbench to test the architecture in real conditions and probe the actual power consumption of the design. The latter was not undertaken during this thesis, due to the lack of material and knowledge at that time. However, the SoC will enable such study in further works. The design is described in details in Appendix 8.2.2. For information, it features a Petalinux OS running on the Cortex ARM A-53 of the board to control and send data to the accelerator in the programmable logic. The SoC features all necessary interfaces to enable direct communication between the Linux user-space and the accelerator (kernel drivers, AXI interface...).

Chapter 4

Synaptic Activity Ratio & Energy Modeling

Chapter contents

This chapter is dedicated to the study of Synaptic Activity in Spiking Neural Networks. Synaptic Activity refers to events occurring in synapses, *i.e.* spikes. As explained earlier in Section 2.4.3.2, Formal Neural Networks (FNN) work in a synchronous fashion with one and only one activation per synapse. On the other hand, the dynamic behavior of SNNs imply an unpredictable number of spikes per synapse and per sample.

The effective number of spikes is crucial when addressing SNN hardware implementations, as it could imply a computation overhead. The computation overhead could counter-balance the advantage of Accumulation (ACC) over Multiplication-Accumulation (MAC) (Section 2.3.1). Hence, the energy savings brought by spiking domain are thus not guaranteed. In this chapter, we study how synaptic activity varies under the influence of dataset, topology and various SNN-specific hyper parameters.

The raw spike count is relevant to compare computational efficiency in different SNN applications or topologies, but it does not help in the comparison with FNNs. Therefore, we propose the Synaptic Activity Ratio (SAR) metric: the ratio between N_{spikes} and N_{synapse}. This metric is inspired from the literature (see Section 2.4.3.2) and is used to assess the computational overhead induced by execution in spiking domain. In doing so, we provide a high-level energy consumption model, based on the relative computation requirements of FNNs and SNNs. The model assimilates the processing of a neuromorphic accelerator to that of its synaptic operations: Multiplication-Accumulation (MAC) for FNNs and Accumulation (ACC) for SNNs. Indeed, Kundu *et. al.* [78] have already shown that computation in neural network accelerators is largely dominated by synaptic operations.

The raw synaptic activity measurements and subsequent SAR results are applied to a benchmark of Machine Learning datasets. Those datasets cover different types of data, with various spatial dimensions and complexities. This study aims in providing a high-level cartography of applications and coding domains.

4.1 Representative Datasets

The seven classification datasets of the benchmark address two types of data: Spatial data and 1D vectors. Each type of data can be mono or multi-channel. There are three datasets for each type of data, seven in total. For each type, there are three datasets with increasing complexity. In this work, state-of-the-art classification accuracy is used as an indicator of complexities. Spatial and vector data are used to evaluate the impact of data type on synaptic activity and SAR.

Image classification datasets are:

1. OPS-SAT RGB [93]: Classification of patches (28x28x3) extracted satellite images in two classes: *cloud* and *no-cloud*. The state-of-the-art accuracy is 81% [93]. This task is part of a larger cloud segmentation application. Figure 4.1 features an example full-size satellite of OPS-SAT dataset on the left. Additionally, the segmentation mask resulting from patch classification is shown on the left.

2. MNIST [95]: Handwritten digits classification task, the most basic and used benchmark application in the SNN community. It is made of ten classes of 28x28 grayscale images. The state-of-the-art accuracy is 99.84% [96]. Some samples of MNIST are shown in Figure 4.2a.

3. GTSRB [97]: German traffic signs classification task, a more complex image classification set with 43 classes of 32x32 RGB images. Its state-of-the-art accuracy is 99.71% [98]. Figure 4.2b presents some examples of GTSRB images.

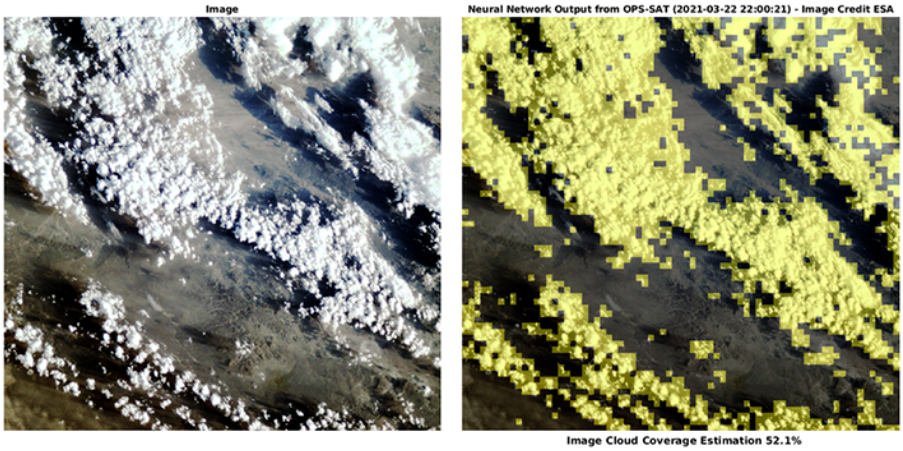


Figure 4.1: Illustration of the CIAR cloud segmentation task. On the left, the original image taken by OPS-SAT. On the right, the resulting segmentation map, with cloudy patches in yellow. Source: [15]

4. CIFAR-10 [99]: common objects classification task, an even more complex image classification task, commonly used in NN community, with 10 classes of 32x32 RGB images. Its state-of-the-art accuracy is 99.70% [100]. A selection of CIFAR-10 samples are shown in Figure 4.3a.

1D data classification datasets are:

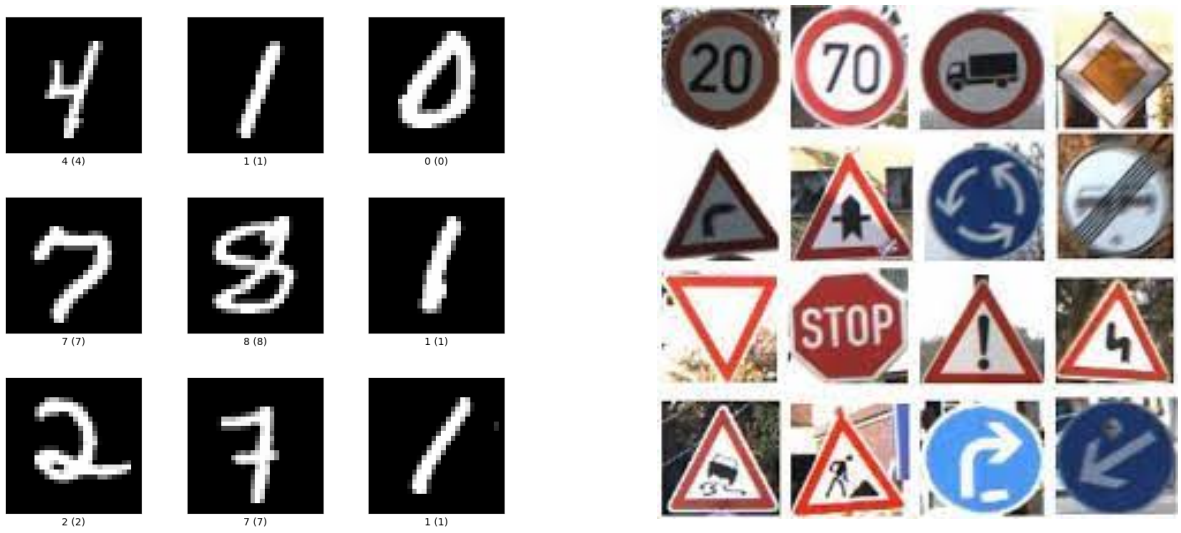
1. Mines versus Rocks [16]: classification of sonar echoes from rocks and military mines in the ground. It is made of two classes of 60-elements vectors, thus it is quite a simple and small classification task. The state-of-the-art accuracy on this task is 90.4% [16]. The task is yet very simple: the input resolution is very low, and there is only two output classes. The low accuracy is mostly due to the low amount of samples in the dataset. Figure 4.3b shows two sonar echoes examples for rocks and mines.

2. Spoken Digits [101]: this dataset is made of the ten groups of 0 to 9 audio digits, extracted from the Google Speech Commands [17] dataset. A Mel Frequency Cepstral Coefficients transform (MFCC) with 39×13 dimensional features was applied to those audio recordings. When this chapter was written (September 2021), no publication mentioning classification accuracy on this dataset was found. However, an accuracy of 97.34% is achieved using the topology given in Table 4.1. Some examples of spectrograms from Google Speech Commands dataset are given in Figure 4.4,

3. Deepsig RadioML 2018 [18]: A dataset which includes both synthetic simulated signals and over-the-air recordings of 24 digital and analog modulation types of RF signals. An illustration of the RF recording experimental setup is given in Figure 4.5. The dataset is made of 24 classes of 1024-elements vectors with 2 channels, and it represents a difficult classification task. Its state-of-the-art classification accuracy is 64% [102].

4.1.1 Used topologies

Each set of training and testing of formal CNNs takes important amount of time (especially with the selected AI framework, i.e. N2D2), so using state-of-the-art deep neural networks would have



(a) (b)

Figure 4.2: a) Sample of MNIST dataset, b) Sample of GTSRB dataset

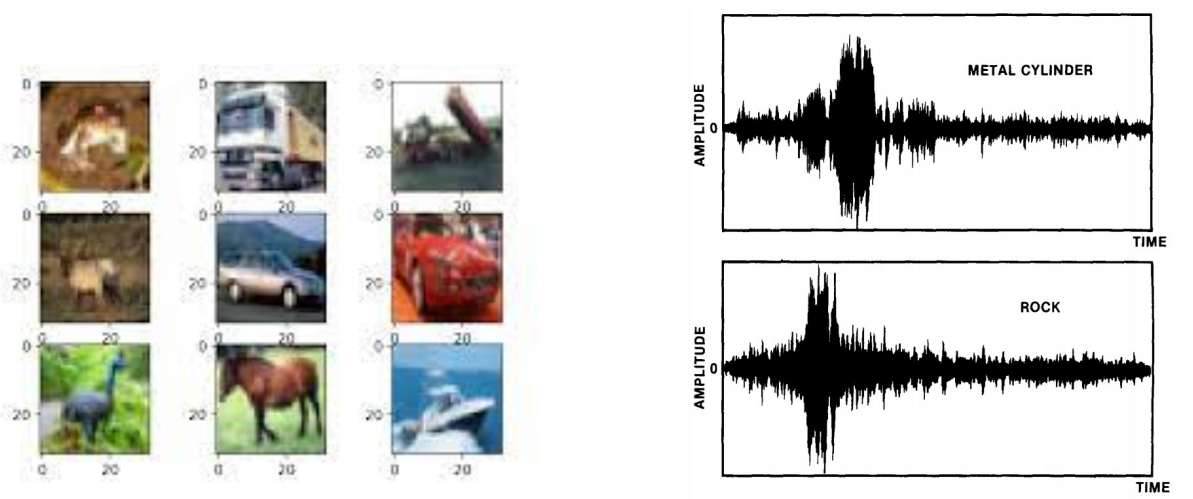
required an unmanageable simulation time. In addition, inference in spike-domain also takes a large amount of time (that is proportional to the complexity and depth of the selected topology). Therefore, we could not use state-of-the-art CNNs to perform this study. However, we made an exploration of topologies and hyper-parameters to provide the best results possible in the limits of our computational power and available time. Instead, a custom CNN was used for each dataset (topology description is available in Tables 4.2). With those topologies, the goal is not to achieve state-of-the-art accuracy, but rather to compare SNNs and FNNs in terms of accuracy and Synaptic Activity Ratio.

4.2 Accuracy and synaptic activity measurements

In this section, we study the impact of dataset and CNN topology on synaptic activity and network accuracy. Additionally, we study the impact of two SNN-specific hyper-parameters: Δ and Θ_{IF}.

4.2.0.0.1 Studying Δ influence Indeed, as explained in Section 2.3.4, this parameter has a strong impact on the behavior of the network: increasing Δ means that the Terminate Delta Module will wait for a larger difference between the most active neuron and the second most active. In other words, the Terminate Delta Module (TDM, Section 3.1.3) will wait for more output spikes. Thus the total number of spikes flowing in the network will also increase. On the other hand, it is usually admitted that a higher Δ will have a positive impact on prediction accuracy [64]. Thus, one of our concerns is to study the possible trade-off between accuracy and synaptic activity.

4.2.0.0.2 Studying Θ_{IF} influence A higher firing threshold in Integrate & Fire (IF) (Θ_{IF}) will cause fewer spikes to pass to the next layer, impacting the overall synaptic activity. However,



(a) (b)

Figure 4.3: a) Sample of CIFAR 10 dataset, b) Sonar echoes from Mines VS Rocks dataset [16]

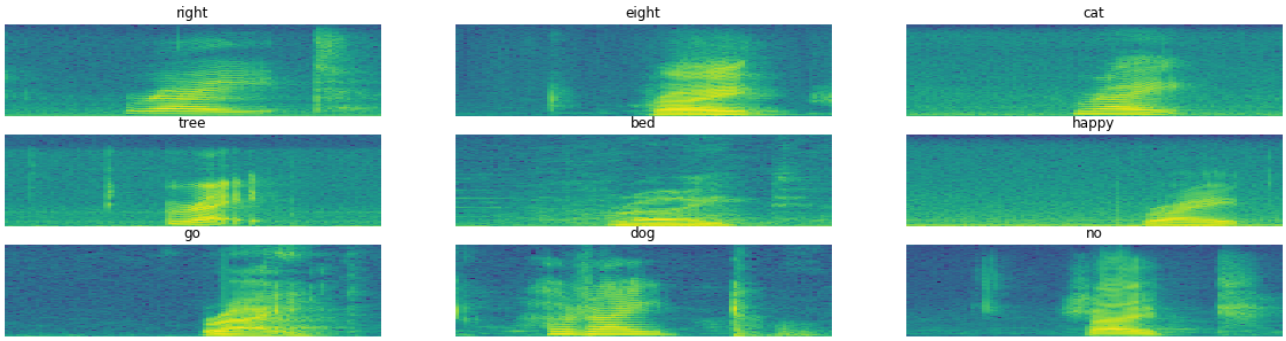


Figure 4.4: Examples of spectrograms from Google Speech Commands dataset [17]. Spoken Digits dataset use in this work is a subset of Google Speech Commands.

it is difficult to predict how the network will react to such a change in Θ_{IF}: the synaptic activity could decrease because fewer spikes are allowed to pass, but the Terminate Delta Procedure could in turn last longer and thus allow more spikes to flow in the network, resulting in an equal number of spikes overall. Hence, studying the influence of this parameter is also one of our concerns. The impact of Θ_{IF} will be assessed on CIFAR-10 only, because its application to all datasets would require unrealistic amount of simulation time.

4.2.1 Methods

A few definitions are provided in the following before going into further details:

task: the combination formed by a dataset and its associated CNN,

experiment: the ensemble formed by the task and the set of hyper-parameters used for training and testing,

trial: is the realization of an experiment (usually, 10 trials are executed for each experiment in order to obtain averaged and statistically representative results).

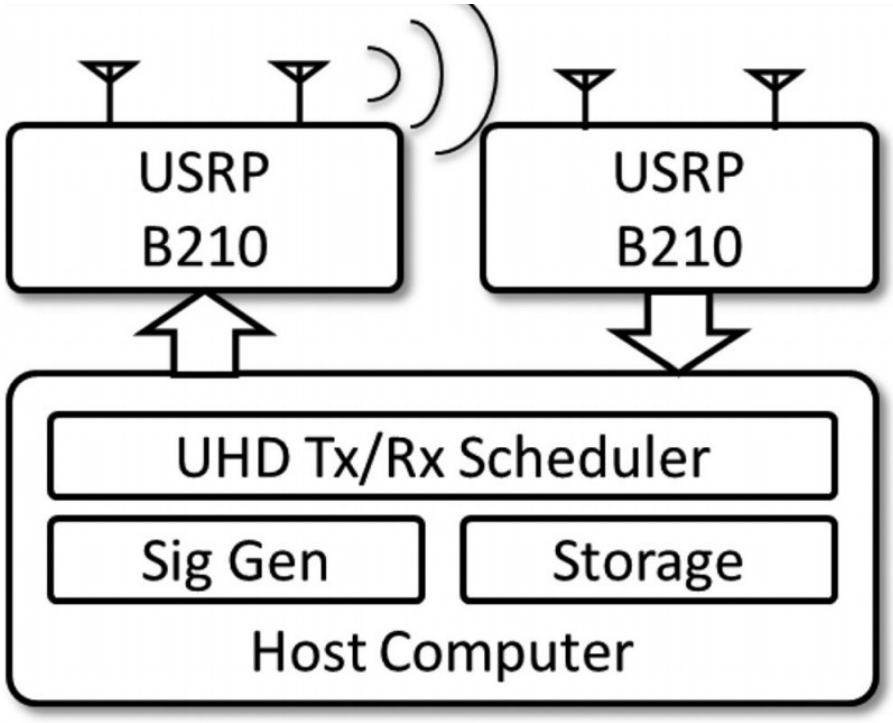


Figure 4.5: Illustration of the over-the-air RadioML 2018 recording setup, found in [18]. A host computer generates the desired RF signal, which is emitted and received via two universal software radio peripherals (USRP).

Each trial is made using N2D2 [1], thus the accuracy and synaptic activity measurements are software simulation results. However, the resulting hardware architecture configured with the trained weights and identical hyper-parameters should reproduce the behavior of N2D2. Consequently, we are confident that the synaptic activity measured in N2D2 will be the same in a subsequent hardware implementation using PADS (Chapter 3) or SPLEAT [14] (Section 2.4.2.8).

4.2.1.0.1 Default hyper-parameters For each trial, N2D2 is configured with the required dataset and its corresponding CNN topology. The hyper-parameters are set to selected values:

- There are no on-the-fly pre-processing on input data;
- $MinPeriod$ is set to 1;
- $MaxPeriod$ is set to 100;
- if not mentioned otherwise, the default value of Δ is 4;
- the default Integrate & Fire threshold Θ_{IF} value is 1.

4.2.1.0.2 Experimental method For each trial:

- the dataset is split with 80% of samples for training, 10% for validation and 10% for testing;

Table 4.1: Large CNN topology for Spoken Digits dataset. This CNN achieves 97.34% accuracy.

Layer	Type	# Kernels	Kernel size	Stride	Output Shape
0	Input				(507,1)
1	Conv	32	5	1	(503,3)
2	MaxPool	3	2	2	(252,3)
3	Conv	64	5	1	(248,5)
4	MaxPool	5	2	2	(124,5)
6	Conv	128	5	1	(120,5)
7	MaxPool	5	2	2	(60,5)
8	FC				250
9	FC				100
10	FC				10

- training is performed until the "learning plateau" is reached, where the network accuracy reaches a stable value and is not increased by further training epochs;
- the trained network is first tested in formal domain, providing a baseline test accuracy; then, the network is transcoded and tested in spiking domain, using N2D2 as well;
- accuracy and synaptic activity are logged during the test (e.g. accuracy of learning and testing, for each of formal and spiking domain);
- Each experiment is averaged on 10 runs.

4.2.2 Synaptic activity results

The experimental protocol is applied to each of the 6 tasks of our benchmark. The results obtained after averaging all measurements on 10 trials are presented in Table 4.3. The synaptic activity is expressed as the number of output spikes of that layer, also called spike count.

4.2.2.0.1 Influence of Δ: When analyzing those results, we observe that the spiking accuracy is always lower than formal accuracy. In other word, transcoding towards spiking domain implies a loss of accuracy. However, this degradation seems to be mitigated by increasing Δ.

Let us imagine a situation where a pixel is encoded in a spike-train with Period = 100. If the Terminate Delta condition is reached before the 100^{th} time-step, the prediction would be enacted before any spike is emitted by this pixel. Thus, the network would have stopped with incomplete information. Increasing Δ implies to wait for more output spikes. In that regard, increasing Δ should result in better accuracy and higher spike counts.

Our results confirms this statement. For example, in the GTSRB task, the $\Delta = 20$ experiment provides an accuracy of 93.5% against 83.3% with $\Delta = 5$. Moreover, the average total spike count is 2.1 times higher with $\Delta = 20$. This trend is found in all other experiments. Thus, tuning Δ enables to optimize the trade-off between accuracy and synaptic activity.

4.2.2.0.2 Influence of $\Theta_{\mathbf{IF}}$: We also study the influence of Θ_{IF} on accuracy and synaptic activity with CIFAR-10 task. The results (Table 4.3) show that, for a constant value of (Δ), higher values Θ_{IF} provide significantly better accuracies. For $\Delta = 5$, the accuracy is 11.45% with

Table 4.2: List of CNN topologies for the classification benchmark

(a) OPSSAT RGB

Layer	Type	# Kernels	Kernel size	Stride	Output Shape
0	Input				(28,28,3)
1	Conv	3	5	1	(24,24,3)
2	MaxPool	3	2	2	(12,12,3)
3	Conv	5	5	1	(8,8,5)
4	MaxPool	5	2	2	(4,4,5)
5	FC				10
6	FC				2

(b) MNIST

Layer	Type	# Kernels	Kernel size	Stride	Output Shape
0	Input				(28,28,1)
1	Conv	6	5	1	(24,24,6)
2	MaxPool	6	2	2	(12,12,6)
3	Conv	16	5	1	(8,8,16)
4	MaxPool	16	2	2	(4,4,16)
5	FC				84
6	FC				10

(c) GTSRB

Layer	Type	# Kernels	Kernel size	Stride	Output Shape
0	Input				(32,32,3)
1	Conv	32	5	1	(28,28,32)
2	MaxPool	32	2	2	(14,14,32)
3	Conv	32	5	1	(10,10,32)
4	MaxPool	32	2	2	(5,5,32)
6	FC				120
7	FC				43
8	FC				10

(d) CIFAR 10

Layer	Type	# Kernels	Kernel size	Stride	Output Shape
0	Input				(32,32,3)
1	Conv	32	5	1	(28,28,32)
2	MaxPool	32	2	2	(14,14,32)
3	Conv	64	5	1	(10,10,64)
4	MaxPool	64	2	2	(5,5,64)
3	Conv	128	5	1	(1,1,128)
6	FC				10

(e) Mines VS Rocks

Layer	Type	# Kernels	Kernel size	Stride	Output Shape
0	Input				(60,1)
1	FC				200
2	FC				2

(f) Spoken Digits

Layer	Type	# Kernels	Kernel size	Stride	Output Shape
0	Input				(507,1)
1	Conv	3	5	1	(503,3)
2	MaxPool	3	2	2	(252,3)
3	Conv	5	5	1	(248,5)
4	MaxPool	5	2	2	(124,5)
6	Conv	5	5	1	(120,5)
7	MaxPool	5	2	2	(60,5)
8	FC				100
9	FC				10

(g) RadioML 2018

Layer	Type	# Kernels	Kernel size	Stride	Output Shape
0	Input				(1024,2)
1	Conv	32	5	1	(1020,32)
2	MaxPool	32	2	2	(510,32)
3	Conv	64	5	1	(506,64)
4	MaxPool	64	2	2	(253,64)
6	Conv	128	5	1	(249,128)
7	MaxPool	128	2	2	(124,128)
8	FC				256
9	FC				125
10	FC				24

$\Theta_{\text{IF}} = 0.5$, 31.8% with $\Theta_{\text{IF}} = 1$ and 57.8% with $\Theta_{\text{IF}} = 2$. As there are 10 classes, the accuracy with $\Theta_{\text{IF}=0.5}$ and $\Delta = 5$ is close to randomness.

However, high values of Θ_{IF} implies higher synaptic activity. For $\Delta = 5$, the total spike count is 1375 with $\Theta_{\text{IF}} = 0.5$, 32967 with $\Theta_{\text{IF}} = 1$ and 313019 with $\Theta_{\text{IF}} = 2$. Those trends are also visible with $\Delta = 10$ and $\Delta = 20$. Thus, Θ_{IF} also enables a trade-off between accuracy and synaptic activity.

However, this behavior is quite surprising. Indeed, lowering Θ_{IF} in a layer means easing output spike firing. Thus, a synaptic activity overhead could be expected for low values of Θ_{IF}. Our hypothesis is that spikes indeed flow more easily through the network with $\Theta_{\text{IF}=0.5}$, including a large number of "undesired" spikes reaching the output. This phenomenon is illustrated in Figure 4.6. This figure shows the spike filtering rate for all 9 experiments on CIFAR-10 task. The filtering rate is the ratio between the numbers of input spikes and output spike as given by Equation 4.1.

Table 4.3: Accuracy and synaptic activity measurements with N2D2 on all datasets of the benchmark

Task	FNN ACC. (%)	Δ	SNN ACC. (%)	Layer 0 (# spikes)	Layer 1 (# spikes)	Layer 2 (# spikes)	Layer 3 (# spikes)	Layer 4 (# spikes)	Layer 5 (# spikes)	Layer 6 (# spikes)	Layer 7 (# spikes)	Layer 8 (# spikes)	Layer 9 (# spikes)	Total (# spikes)
OPS-SAT (Std. dev.)	**91.95 (0.37)**	5	**91.78 (0.52)**	200 (84)	216 (60)	104 (25)	79 (20)	37 (6)	8 (1)	4 (0)				648 (196)
		10	**91.88 (0.83)**	683 (252)	561 (116)	243 (38)	228 (36)	85 (9)	18 (2)	10 (0)				1828 (453)
		20	**91.92 (1.21)**	9 117 (14 351)	1 057 (477)	394 (136)	379 (141)	114 (32)	26 (11)	13 (3)				11 100 (15 151)
MNIST (Std. dev.)	**99,2 (0,2)**	5	**97,3 (0,6)**	128 (12)	870 (56)	391 (23)	481 (38)	269 (19)	65 (5)	38 (3)				2242 (156)
		10	**98,9 (0,5)**	263 (16)	2 525 (106)	1 028 (34)	895 (94)	501 (45)	80 (4)	55 (2)				5 347 (301)
		20	**99,1 (0,6)**	501 (22)	3 921 (188)	1 570 (60)	1 388 (144)	768 (69)	141 (5)	98 (3)				8 387 (491)
GTSRB (Std. dev.)	**95,5 (0,8)**	5	**83,3 (5,1)**	4 083 (549)	8 195 (1 373)	3 052 (503)	2 054 (448)	1 231 (243)	145 (19)	136 (14)	175 (21)			19 071 (3 170)
		10	**90,6 (1,8)**	6 285 (937)	10 852 (2 583)	3 979 (927)	3 260 (690)	1 936 (391)	259 (53)	272 (42)	366 (58)			27 209 (5 681)
		20	**93,5 (1,4)**	8 947 (1 239)	15 888 (3 167)	5 779 (1 112)	4 574 (948)	2 738 (547)	435 (76)	481 (64)	646 (91)			39 488 (7 244)
CIFAR-10 (Std. dev.)	**69,8 (1,1)**	5 $\Theta_{IF} = 1$	**31,8 (6,1)**	4 772 (953)	16 167 (2 281)	6 275 (803)	3 395 (519)	2 129 (254)	87 (9)	82 (8)				32 967 (4 837)
		10 $\Theta_{IF} = 1$	**50,2 (2,7)**	10 655 (591)	30 490 (2 482)	11 446 (846)	7 169 (775)	4 479 (414)	220 (9	201 (8)				64 660 (5 125)
		20 $\Theta_{IF} = 1$	**57,8 (1,9)**	22 919 (771)	58 790 (4 786)	22 091 (1 591)	15 943 (1 668)	10 074 (932)	531 (21)	482 (20)				130 830 (9 789)
		5 $\Theta_{IF} = 2$	**59,2 (0,9)**	56 165 (1 592)	65 231 (6 567)	24 978 (2 134)	9 298 (1 244)	6 005 (667)	161 (12)	60 (3)				161 898 (12 409)
		10 $\Theta_{IF} = 2$	**59,2 (0,9)**	83 542 (1 740)	97 492 (10 141)	37 323 (3 213)	141 341 (1 881)	9 225 (1 006)	254 (19)	98 (4)				242 275 (18 004)
		20 $\Theta_{IF} = 2$	**59,2 (0,9)**	110 520 (1 522)	120 233 (13 565)	49 826 (4 322)	19 464 (2 506)	12 484 (1 341)	352 (37)	140 (7)				313 019
		5 $\Theta_{IF} = 0.5$	**11.45 (0.002)**	5 (1)	104 (14)	64 (8)	603 (75)	388 (43)	68 (3)	143 (6)				1376 (150)
		10 $\Theta_{IF} = 0.5$	**11.54 (0.002)**	68 (15)	1 310 (193)	673 (82)	2 780 (322)	1 617 (161)	137 (6)	316 (13)				6 900 (792)
		20 $\Theta_{IF} = 0.5$	**12.14 (0.008)**	574 (185)	7 714 (1 314)	3 401 (480)	6 713 (619)	3 797 (268)	257 (12)	656 (39)				23 112 (2 916)
Mines VS Rocks (Std. dev.)	**79,2 (9,8)**	5	**74,2 (5,8)**	136 (69)	299 (159)	8 (2)								443 (230)
		10	**75 (4,4)**	259 (72)	576 (158)	20 (8)								855 (238)
		20	**78,8 (5,6)**	490 (232)	1 090 (499)	34 (24)								1 614 (755)
Spoken Digits (Std. dev.)	**88,55 (2,3)**	5	**38,2 (10,1)**	13 918 (1 684)	24 908 (17 681)	13 348 (8 975)	4 839 (772)	3 223 (561)	1 013 (280)	884 (243)	93 (3)	43 (3)		61 473 (30 202)
		10	**71,6 (4,7)**	103 463 (17 488)	5 776 (3 534)	3 148 (1 914)	2 899 (736)	1 891 (496)	1 839 (560)	1 367 (422)	221 (16)	308 (17)		120 912 (25 083)
		20	**87,5 (4,2)**	228 752 (26 123)	12 988 (8 818)	6 933 (4 481)	5 727 (948)	5 705 (682)	3 068 (932)	2 351 (708)	387 (28)	578 (57)		244 128 (42 777)
RadioML (Std. dev.)	**42 (0,3)**	5	**20,5 (5,6)**	16 791 (3 542)	108 234 (23 076)	64 131 (13 455)	12 5167 (24 690)	86 702 (16 732)	13 201 (1 936)	12 185 (1 846)	191 (19)	119 (15)	265 (34)	42 6986 (85 345)
		10	**31,6 (4,9)**	32 167 (2 479)	206 629 (26 715)	121 349 (15 074)	232 232 (25 878)	160 043 (16 370)	23 934 (2 201)	22 191 (2 058)	471 (54)	302 (26)	714 (68)	800 033 (90 923)
		20	**37,5 (3,8)**	508 836 (1 290)	325 052 (36 528)	190 171 (20 596)	360 255 (30 068)	248 090 (18 679)	38 796 (2 584)	36 050 (2 412)	895 (75)	595 (37)	1442 (83)	98 4637 (112 352

$$\mathrm{FR}(L_i, L_j) = \frac{N^i}{N^j} \tag{4.1}$$

Where $\mathrm{FR}(L_i, L_j)$ is the filtering rate from layer i to layer j and N^i the spike count at layer i.

For $\Theta_{\mathrm{IF}=1}$ and $\Theta_{\mathrm{IF}=2}$, the filtering rate is always larger than one, thus there is always more input spikes than output spikes: the SNN acts as a spike (low-pass) filter. $\Theta_{\mathrm{IF}=0.5}$ experiments demonstrate the opposite: there are more output spikes than input spikes. This observation backs our hypothesis that a low threshold generates "undesired" spikes in the intermediate layers. This phenomenon is also responsible for the drastically deteriorated accuracy, as the Terminate Delta procedure is altered by undesired spikes.

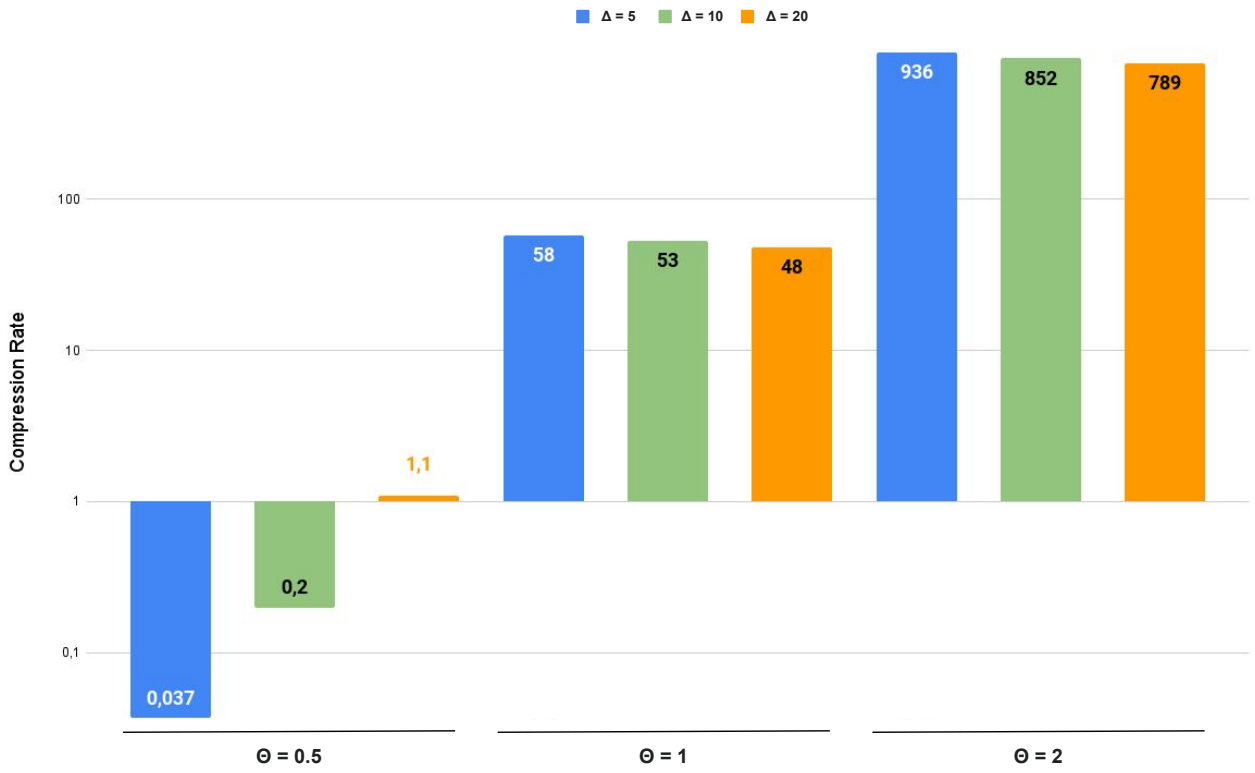


Figure 4.6: Evolution of spike filtering rate on CIFAR-10 task

4.2.2.0.3 Combined effect of Δ and Θ_{IF} Moreover, Δ and Θ_{IF} have a combined effect on accuracy and synaptic activity. We have seen that Δ had an impact on accuracy in GTSRB and other tasks. This is also the case on CIFAR-10 with $\Theta_{\text{IF}} = 0.5$ and $\Theta_{\text{IF}} = 1$. But with $\Theta_{\text{IF}} = 2$, the accuracy is constant for all values of Δ. However, the synaptic activity is still affected. With $\Theta_{\text{IF}} = 2$, the total synaptic activity for $\Delta = 5$ is 161898, 242275 for $\Delta = 10$ and 313019 for $\Delta = 20$. In order to further investigate the combined effect of Δ and Θ_{IF}, we analyze the Spike Count Evolution (SCE) between $\Delta = 5$ and $\Delta = 20$ against Θ_{IF}, as described in Equation 4.2.

$$\text{SCE}^l(\Delta_1, \Delta_2) = \frac{N^l_{\Delta_1}}{N^l_{\Delta_2}} \tag{4.2}$$

Where $\text{SCE}^l(\Delta_1, \Delta_2)$ is the SCE of layer l between Δ_1 and Δ_2, and $N^l_{\Delta_1}$ the spike count at layer l for Δ_1.

The SCE is measured in the input layer (Layer #0), in the output layer (Layer #6) and in total. The results are given in Figure 4.7. The vertical axis is logarithmic. The general trend is that SCE decreases with Θ_{IF}, *i.e.* the spike count is more affected by Δ with low Θ_{IF} values. Moreover, for $\Theta_{\text{IF}} = 1$ and $\Theta_{\text{IF}} = 2$, the SCE is higher for the output layer than for the input. This trend is drastically reversed when addressing $\Theta_{\text{IF}} = 0.5$: the SCE is more than 50 times higher in the input layer than in the output layer. If we refer to Figure 4.6, we can thus assume that the evolution of filtering rate is mostly due to the SCE in the input layer. In other words, the SNN behavior is deteriorated by the excessive number of spikes in the first layer.

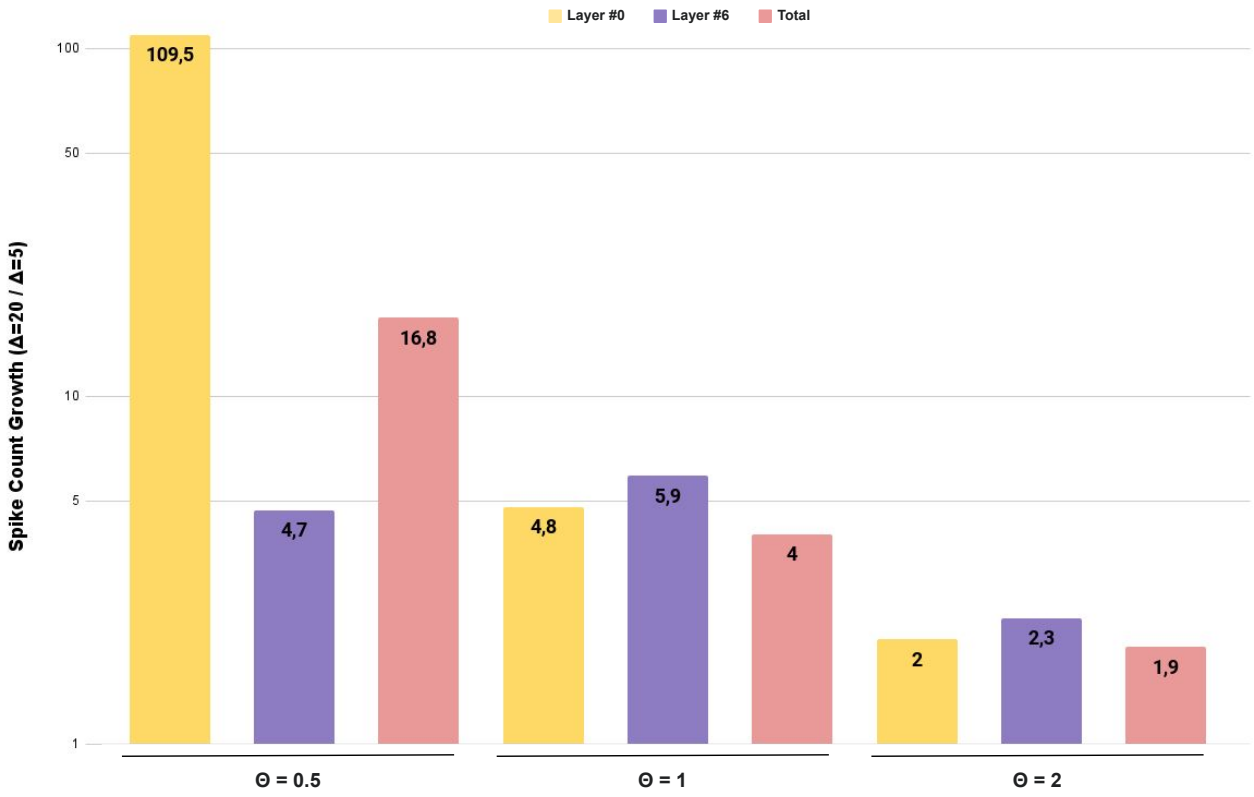


Figure 4.7: Spike Count Evolution $\Delta = 5$ and $\Delta = 20$ in Layer #0, Layer #6 and in total, for various Θ_{IF} on CIFAR-10 task. The vertical axis is logarithmic

To prevent this behavior, Θ_{IF} could be decreased locally in the first layer, as proposed in the Spike Select method proposed in [64]. However, preliminary experiments showed that accuracy was drastically affected by the Spike Select technique, thus it was let out of scope of our study. Further work is therefore required in that regard.

4.2.2.0.4 Influence of dataset At first glance, the synaptic activity measurements are indeed very different from one task to another. However, the number of synapses involved has an impact on this metric. To isolate the intrinsic effect of dataset, synaptic activity can be normalized with respect to number of synapses. This metric is proposed in the next section: Synaptic Activity Ratio (SAR).

4.2.3 Discussions on Synaptic Activity results

In this section, we have studied the influence of the Terminate Delta criterion (Δ) and the Integrate & Fire neuron threshold (Θ_{IF}) on synaptic activity and accuracy of spiking CNNs on various datasets. Both SNN-specific hyper-parameters highlight the existing trade-off between synaptic activity and accuracy.

Increasing the value of Δ improves accuracy at the expense of a greater synaptic activity. Similarly, increasing the value of Θ_{IF} also improves accuracy and increases synaptic activity. Moreover, the filtering rate of the network is severely affected by Θ_{IF}. With $\Theta_{\text{IF}=0.5}$, there are

more output spikes than input spikes. On the other hand, with $\Theta_{\text{IF}} = 1$ and $\Theta_{\text{IF}} = 2$ we observe the opposite: the network acts as a strong spike filter, with fewer output spikes at the output than at the input.

Additionally, some combined effects of Δ and Θ_{IF} have been found. If Delta has a strong influence on accuracy with $\Theta_{\text{IF}} = 0.5$ and $\Theta_{\text{IF}} = 1$, this is not the case when $\Theta_{\text{IF}} = 2$: the accuracy remains the same for all values of Δ. Moreover, as explained above, the influence of Δ on the filtering rate varies with Θ_{IF}: for $\Theta_{\text{IF}} = 0.5$ the filtering rate increases with respect to Δ, but we observe the opposite for $\Theta_{\text{IF}} = 1$ and $\Theta_{\text{IF}} = 2$. Lastly, for $\Theta_{\text{IF}} = 0.5$, the SCE was greater in the input layer. On the other hand, the SCE was greater in the output layer for $\Theta_{\text{IF}} = 1$ and $\Theta_{\text{IF}} = 2$.

Consequently, this study helps to understand the behavior of a rate-coded Spiking Neural Network, and how it reacts to different hyper-parameters. More specifically, the study highlighted a strong trade-off between accuracy and synaptic activity. As discussed in our experiments on PADS architecture (Section 3.2), higher synaptic activity implies higher signal toggle rate in the design. In turns, an higher toggle rate implies higher dynamic power usage. Thus, there exists an interesting trade-off between power usage and accuracy. The SAR eases the exploration of such trade-off, as it only requires software-level information.

However, those results only enable the comparison between various experiments on a same task. As explained previously, the spike count has to be normalized to the number of synapse in order to evaluate the impact of the dataset independently from the topology. Moreover, one of our main concern is comparing SNNs and FNNs. Hence, we introduce a metric called "Synaptic Activity Ratio" in the next section. Indeed, in addition to enabling comparison between different SNN topologies, this metric could also enable high-level comparison of SNNs and FNNs in terms of number of operations, and thus, in terms of relative energy consumption per sample.

Lastly, the investigation was limited to converted rate-coded SNNs. The results could be different using other spike-coding techniques, advanced weight and threshold balancing methods, or even totally different SNN model representation. However, the Synaptic Activity Ratio metric proposed in the next section is designed to be adapted to any SNN model.

4.3 Synaptic Activity ratio

The Synaptic Activity Ratio (SAR) is a novel metric inspired from previous works, which proposes to count operations in SNNs and equivalent FNNs to assess their relative energy efficiency. The number of spikes per synapse was proposed in various forms in the literature [89] [31] [78] to assess potential energy gains of SNNs. This metric remains however qualitative and lacks formalization. In this section, we propose a formalization through the SAR. Moreover, we develop an high-level model for energy-wise cartography of applications and coding domains that is based on the SAR metric. This model will be confronted to actual energy estimations in Chapter 5. It should be noted that the model is only adapted to FPGA implementations.

4.3.1 Energy Consumption Model

As a reminder, we call synaptic operation the integration of an input activation in the neuron. In both FNNs and SNNs, the computation is widely dominated by synaptic operations [78] in Convolution and Fully-Connected layers. This observations comes from measurements of Floating Point Operation per second (FLOP/s) and energy in SNNs and ANNs is directly proportional with

the number of synapses remaining after pruning, that is the number of ACC and MAC operations respectively. Hence, all other computations are neglected in our model. Consequently, the energy consumption of a NN model can be expressed as the energy consumption of its synaptic operations, as shown in Equation 4.3

$$\begin{aligned} E_{\text{FNN}} &= N_{\text{MAC}} * E_{\text{MAC}} \\ E_{\text{SNN}} &= N_{\text{ACC}} * E_{\text{ACC}} \end{aligned} \tag{4.3}$$

Where E_{FNN} (respectively E_{SNN}) is the energy consumption of a formal (respectively spiking) neural network implementation, and E_{MAC} (respectively E_{ACC}) is the energy cost of a MAC (respectively ACC) operation.

In this model, the execution time of the architecture is assimilated to the execution time of synaptic operations (MAC & ACC). The rest of the computation is neglected just as for energy consumption, since synaptic operations are largely dominant in neural network processing.

Let $\lambda = \frac{E_{\text{MAC}}}{E_{\text{ACC}}}$ be the ratio between the energy consumptions of a MAC and an ACC operation. The value of λ depends on the implementation technology of each neuron (digital or analog, using LUTs/FFs or dedicated DSPs, semiconductor technology etc.). For example, in the literature we found that Panda *et. al.* [76], working with 32nm CMOS technology and 32-bit dynamics, measured $3.2pJ$ energy consumption for a MAC, and $0.1pJ$ for an ACC. Hence, and for that specific case, $\lambda = 32$. Integrating λ into Equation 4.3 yields to Equation 4.4, which defines the relative energy model between spiking and formal domains.

$$\frac{E_{\text{SNN}}}{E_{\text{FNN}}} = \frac{1}{\lambda} \times \frac{N_{\text{ACC}}}{N_{\text{MAC}}} \tag{4.4}$$

In a Formal Neural Network, there is one single activation per synapse, as information is static and synchronous. Therefore, there is one and only one MAC operation per synapse. Thus $N_{\text{MAC}} = N_{\text{synapses}}$. On the other hand, in SNNs the number of activation is the number of spikes. Synaptic activity is non-deterministic, but depends on several hyper-parameters (Section 4.2) and random initialization. Hence, $N_{\text{ACC}} = N_{\text{spikes}}$. Replacing this equality in Equation 4.4 yields to Equation 4.5.

$$\begin{aligned} \frac{E_{\text{SNN}}}{E_{\text{FNN}}} &= \frac{1}{\lambda} \times \frac{N_{\text{spikes}}}{N_{\text{synapses}}} \\ \frac{E_{\text{SNN}}}{E_{\text{FNN}}} &= \frac{1}{\lambda} \times \text{SAR} \end{aligned} \tag{4.5}$$

Indeed, SAR is defined as the ratio between the spike count and the number of synapses. Additionally, Equation 4.6 represents an alternative SAR metric defined at layer level. This layer-level SAR is a fully novel proposition, whose aim is to study the SAR metric at a finer granularity. In details, layer-level SAR can be expressed using the output spike count from the previous layer, enabling easy computation based on the N2D2 logs. It should be noted that this layer-wise SAR only makes sense for Convolution or Fully-Connected layers. Pooling layers are not covered as computation in the network is dominated by Convolution and Full-Connected layers [78].

$$
\begin{aligned}
\mathrm{SAR}^l &= \frac{N^l_{\mathrm{ACC}}}{N^l_{\mathrm{MAC}}} \\
\mathrm{SAR}^l &= \frac{N^l_{\text{input spikes}}}{N^l_{\text{input synapses}}} \\
\mathrm{SAR}^l &= \begin{cases} \frac{N^{l-1}_{\text{output spikes}} * N^l_{\text{neurons}}}{N^{l-1}_{\text{neurons}} * N^l_{\text{neurons}}} \text{ if l is FC} \\ \frac{N^{l-1}_{\text{output spikes}} * N^l_{\text{filters}} * (F^l_{\text{width}} - S^l)^2}{N^{l-1}_{\text{neurons}} * N^l_{\text{filters}} * (F^l_{\text{width}} - S^l)^2} \text{ if l is Conv} \end{cases}, \\
\mathrm{SAR}^l &= \frac{N^{l-1}_{\text{output spikes}}}{N^{l-1}_{\text{neurons}}}
\end{aligned}
\tag{4.6}
$$

With:

- SAR^l the Synaptic Activity Ratio for layer l,
- N^l_{ACC} the number of ACC for spiking layer l and N^l_{MAC} the number of MAC for its formal counterpart,
- $N^l_{\text{input spikes}}$ the number of input spikes for a layer l for an inference on a single sample,
- $N^l_{\text{input synapses}}$ the number of input synapses of a layer l
- N^l_{neurons} the number of output neurons of a layer l,
- N^l_{filters} the number of filters, F^l_{width} the width of the filter and S_l the stride of a convolution layer l.

4.3.2 The value of λ

In this subsection, we focus on the value of λ, the energy consumption ratio between MAC and ACC operations. More specifically, we aim at finding the λ value corresponding to our specific hardware target: Xilinx xc7z020 FPGA (Zedboard). The λ value will be useful when interpreting the SAR results. Indeed, according to Equation 4.5, λ provides a frontier between coding domains: if SAR$< \lambda$, $E_{\mathrm{SNN}} < E_{\mathrm{FNN}}$ and *vice versa*.

In FPGAs, operations are usually mapped to the programmable logic, in LUTs and Flip-Flops. However, Xilinx boards feature specific circuits for Digital Signal Processing: DSPs. Those processing units are optimized for multiplication operation. During hardware synthesis, Vivado maps MAC operations on DSPs. However, DSP is a scarce resource on FPGAs: there are only 220 DSPs on a Zedboard, and 2520 on a ZCU102. To cope with this scarcity, Vivado is able to map several parallel MAC operations on a single DSP by increasing its working frequency. Indeed, DSPs have a maximum working frequency of 741MHz [103]. Usually, we work with a clock frequency of 100MHz, thus a single DSP can be shared for up to 7 parallel MAC operations. In all, Vivado is able to synthesize up to 1540 different MAC operations on the 220 available DSPs of a Zedboard. We thus have the following statement for Zedboard:

- For 0 to 1540 MAC operations, Vivado uses DSPs

Table 4.4: Logic resources, power and energy comparison for 16-bit ACC and MAC operations on Xilinx xc7z020 FPGA. The MAC operation has been implemented with and without DSPs.

	ACC	**MAC w/o DSP**	**MAC w DSP**
LUT (#)	16	253	16
FF (#)	80	112	48
DSP (#)	0	0	1
Dynamic Power (mW)	1.6	7.1	1.8
Energy (pJ)	16	71	18

- Above 1540 MAC operations, Vivado uses LUTs and FFs.

In order to characterize the λ value, we have measured the hardware footprint of ACC operation as well as MAC operation with and without using DSPs on Xilinx Zedboard (Table 4.4). In this table, we have $\lambda_1 = \frac{\text{Energy}_{\text{MAC DSP}}}{\text{Energy}_{\text{ACC}}} = 1.1$ and $\lambda_2 = \frac{\text{Energy}_{\text{MAC no DSP}}}{\text{Energy}_{\text{ACC}}} = 4.4$. When DSPs are used the ratio is named λ_1, and λ_2 otherwise. Therefore, the value of λ for a Xilinx Zedboard is defined as shown in Equation 4.7.

$$\lambda = \begin{cases} 1.1 \text{ if } N_{\text{MAC}} \leq 1400 \\ \frac{1.1 \times 1400 + 4.4 \times (N_{\text{MAC}} - 1400)}{N_{\text{MAC}}} \text{ if } N_{\text{MAC}} \geq 1400 \end{cases} \tag{4.7}$$

Where N_{MAC} is the number of parallel MAC operations.

Furthermore, Equation 4.7 can be generalized to other Xilinx boards, as shown in Equation 4.8.

$$\lambda = \begin{cases} \lambda_1 \text{ if } N_{\text{MAC}} \leq 7 \times N_{\text{MAC}}^{\text{max}} \\ \frac{\lambda_1 \times 7 \times N_{\text{MAC}}^{\text{max}} + \lambda_2 \times (N_{\text{MAC}} - 7 \times N_{\text{MAC}}^{\text{max}})}{N_{\text{MAC}}} \text{ if } N_{\text{MAC}} \geq 7 \times N_{\text{MAC}}^{\text{max}} \end{cases} \tag{4.8}$$

Where N_{MAC} is the number of parallel MAC operations, $N_{\text{MAC}}^{\text{max}}$ is the number of available DSPs on the board, λ_1 is the energy consumption ($\frac{E_{\text{MAC}}}{E_{\text{ACC}}}$) using DSPs and λ_2 without using DSPs.

For ZCU102, we measure the energy of ACC and MAC operations with and without DSPs:$E_{\text{ACC}}^{\text{ZCU102}} = 18fJ$, $E_{\text{MAC w\ DSP}}^{\text{ZCU102}} = 27fJ$ and $E_{\text{MAC w\o DSP}}^{\text{ZCU102}} = 103fJ$. Thus, we found $\lambda_1^{\text{ZCU10Z}} = 1.5$, and $\lambda_2^{\text{ZCU10Z}} = 5.7$. Moreover, in the case of ZCU102, there are 2520 available DSPs, thus $N_{\text{MAC}}^{\text{max}} = 2520$. Using Equations 4.7 and 4.8, we compute λ for Zedboard and ZCU102, with respect to the number of synapses in the design (*i.e.* the number of MAC operations). The results are provided in Figure 4.8.

In order to estimate λ for Zedboard (λ_{ZE}) and ZCU102 (λ_{ZC}), we count the number of MACs in VGT [5] architecture with respect to the network topology. Consequently, the SAR metric is specifically adapted to a comparison between PADS and VGT, two equivalent parallel neuromorphic accelerators. Equations 4.9 are derived from the VGT code. The code is available on

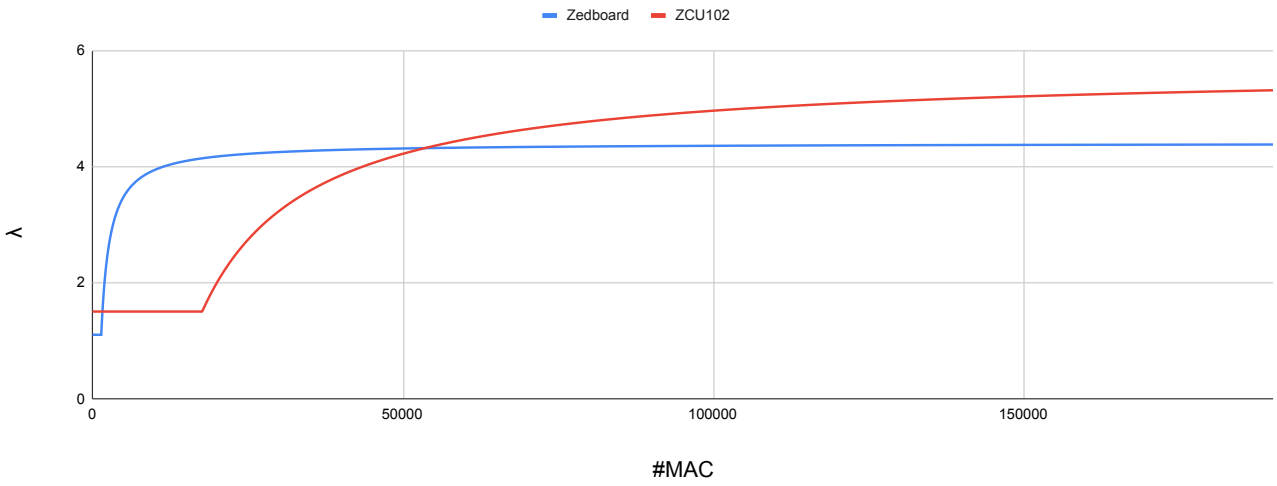


Figure 4.8: Evolution of λ against number of parallel MACs, for Xilinx Zedboard and ZCU102 targets.

GitHub [104]. Those equations enable to compute the number of MACs required for Conv and FC layers in VGT architecture.

$$\begin{aligned} N_{\text{Conv}}^{\text{MAC}} &= N_{\text{In}}^{\text{Chan}} \times N_{\text{Out}}^{\text{Chan}} \times H_{\text{Filter}} \times W_{\text{Filter}} \\ N_{\text{FC}}^{\text{MAC}} &= N_{\text{In}}^{\text{Neur}} \times N_{\text{Out}}^{\text{Neur}} \end{aligned} \tag{4.9}$$

Where $N_{\text{Conv}}^{\text{MAC}}$ is the number of MACs in a VGT convolution layer with $N_{\text{In}}^{\text{Chan}}$ input and $N_{\text{Out}}^{\text{Chan}}$ output channels, using a convolution filter of height H_{Filter} and width W_{Filter}. $N_{\text{FC}}^{\text{MAC}}$ is for Fully-Connected layers having $N_{\text{In}}^{\text{Neur}}$ input and $N_{\text{Out}}^{\text{Neur}}$ output neurons.

We estimate the total number of MACs for each topology using Equation 4.9. The subsequent counts are shown in Table 4.5a. In the end, the values of λ_{ZE} and λ_{ZC} given in Table 4.5a are derived from Equation 4.5 based on the number of MACs shown in Table 4.5a. The resulting λ values are used in the next section. It helps determining if an application is suitable for spiking domain regarding SAR: if SAR$< \lambda$, $E_{\text{SNN}} < E_{\text{FNN}}$ and *vice versa*.

4.4 Synaptic Activity Ratio measurements

Based on the synaptic activity results of Section 4.2 (Table 4.3), we compute the layer-wise and network-wise SAR for each experiment. To do so, a python software is developed. The program takes synaptic activity trace from N2D2 and computes layer-wise and network-wise SAR.

4.4.1 Synaptic Activity Ratio evaluation software

The SAR estimation software works on N2D2 output files: synaptic activity logs. Those logs contains the synaptic trace of all layers of the network for one image. A dedicated python plug-in catches those files on the fly as they are automatically overwritten by N2D2 at each log. For each layer, 50 activity logs are stored in a auto-generated folder tree. The software then translates

(a) Number of MAC operations in the VGT architecture adapted to each CNN of our benchmark (topologies in Tables 4.2).

Task	Number of MACs
MNIST	24 894
GTSRB	137 682
CIFAR-10	260 680
Mines VS Rocks	12 400
Spoken Digits	137 692
RadioML 2018	40 722 383

(b) Estimated values of λ on the dataset benchmark using VGT architecture on Zedboard and ZCU102

Task	λ_{ZE}	λ_{ZC}
OPSSAT	1.5	1.35
MNIST	4.21	2.72
GTSRB	4.37	5.15
CIFAR-10	4.38	5.42
Mines VS Rocks	4.03	1.50
Spoken Digits	4.25	3.32
RadioML 2018	4.40	5.70

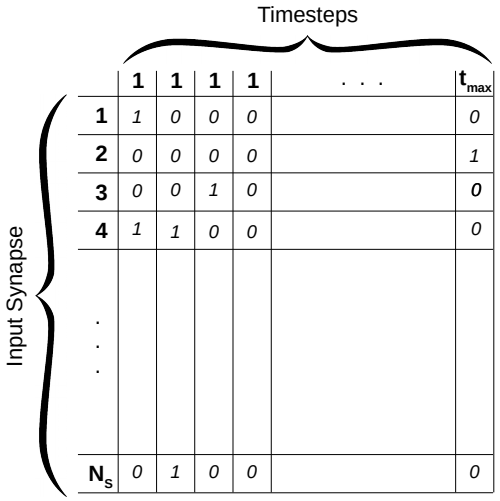


Figure 4.9: Unified synaptic trace format

the activity logs into a unified format: a 2D matrix with flattened input synapses as rows and timesteps as columns. This format is shown in Figure 4.9. Each cell contains the state of an input synapse i at a given timestep: 1 if it carries a spike and 0 otherwise.

To build this unified synaptic trace format, the real-time synaptic trace of N2D2 is discretized. The temporal sampling rate (*timestep length*) is set according to the real PADS GenCell behavior (see Section 3.1.1). A function automatically scans all input synaptic traces and determines the minimum delay between two spikes δt. As the GenCell emits at most one spike per timestep, the temporal resolution of the discretization process is set to δt. As a reminder, the timestep duration of PADS GenCell is equal to the input size in clock cycles (784 clock cycles for MNIST, 3072 for GTSRB...). The program then counts the total number of spikes per synapse and computes the SAR according to formulas 4.5 and 4.6 for network-wise and layer-wise SAR respectively. Layer-wise and network-wise SAR are computed for each experiment of the benchmark using the dynamic SAR estimator software.

Table 4.6: Large CNN topology for MNIST dataset

Layer	Type	# Kernels	Kernel size	Stride	Output Shape
0	Input				(28,28,1)
1	Conv	32	5	1	(24,24,32)
2	MaxPool	32	2	2	(12,12,32)
3	Conv	32	5	1	(8,8,32)
4	MaxPool	32	2	2	(4,4,32)
6	FC				120
7	FC				84
8	FC				10

4.4.2 Network-wise SAR & theoretical cartography

The SAR results are presented in Figures 4.10 and 4.11. In those figures, C_i stands for the i^{th} convolution layer, and FC_i stands for the i^{th} fully-connected layer. In this section, we analyze the network-wise SAR results (in gray) on the dataset benchmark. These values are compared to λ_{ZE} and λ_{ZC} to assess the suitability to spiking domain. The network-wise results are compiled in Table 4.7 . In this table, the cells are green when the SAR is above λ_{ZE} and λ_{ZC}. This is the best case for spiking domain according to Equation 4.5. The cells are yellow when SAR is between the two λ values. In this case, the suitability depends on the hardware target. When SAR is below both λ values, the case is favorable to formal domain and cells are in red.

The network-wise SAR increases with Δ in each topology, as expected from the raw synaptic activity analysis. Moreover, the SAR seems to increase with task complexity. The SAR is indeed higher in CIFAR-10 than in GTSRB, which is higher than in MNIST. The same trend is visible for vector data (Mines VS Rocks, Spoken Digits and RadioML 2018). In other words, spiking acceleration seems better suited for simple tasks, at least for rate-based models. Additionally, the SAR is generally higher for 1D vector datasets at comparable complexity level. For both Spoken Digits and RadioML datasets, the best case accuracy ($\Delta = 20$) fails to meat the λ. According to the proposed model, the SNN will thus use more energy than an equivalent FNN on those tasks. More generally, the SAR results tend to indicate that spiking domain is more suited to image datasets than 1D vector datasets. It should be noted that this statement is constrained to converted rate-coded SNNs and parallel accelerators. When analyzing those results in the light of the lambda values of Table 4.5b, we observe that spiking domain is preferable in environments where DSPs are saturated. Indeed, λ increases when DSPs are saturated (Figure 4.8). This is visible on the figures, as λ_{ZE} is usually lower than λ_{ZC}. As large topology uses more MACs, and thus more DSPs, this statement should imply that spiking domain takes advantage from topology size. For example, OPS-SAT is a very small CNN topology, thus the values of λ are small. Thus, OPS-SAT with $\Delta = 20$ is not suitable to spiking domain even if the SAR is low (2.2). However, topologies associated to complex datasets are larger than that of simple datasets. In order to separate the effect of topology size and dataset complexity, we use a larger CNN topology on MNIST and measure SAR with N2D2. The larger topology is provided in Table 4.6. The subsequent SAR measurements and lambda values are shown in Figure 4.12.

The SAR for MNIST with either the baseline (Figure 4.10) or larger topology (Figure 4.12) are very close. The values of λ_{ZE} and λ_{ZC} are much greater in the larger CNN ($\lambda_{\text{ZE}}^{\text{OPSSAT}} = 1.5$ and $\lambda_{\text{ZE}}^{\text{RadioML}} = 4.40$), thus this implementation is even more favorable to spiking domain, according to the SAR energy model. This experiment thus comforts our hypothesis that SAR is related

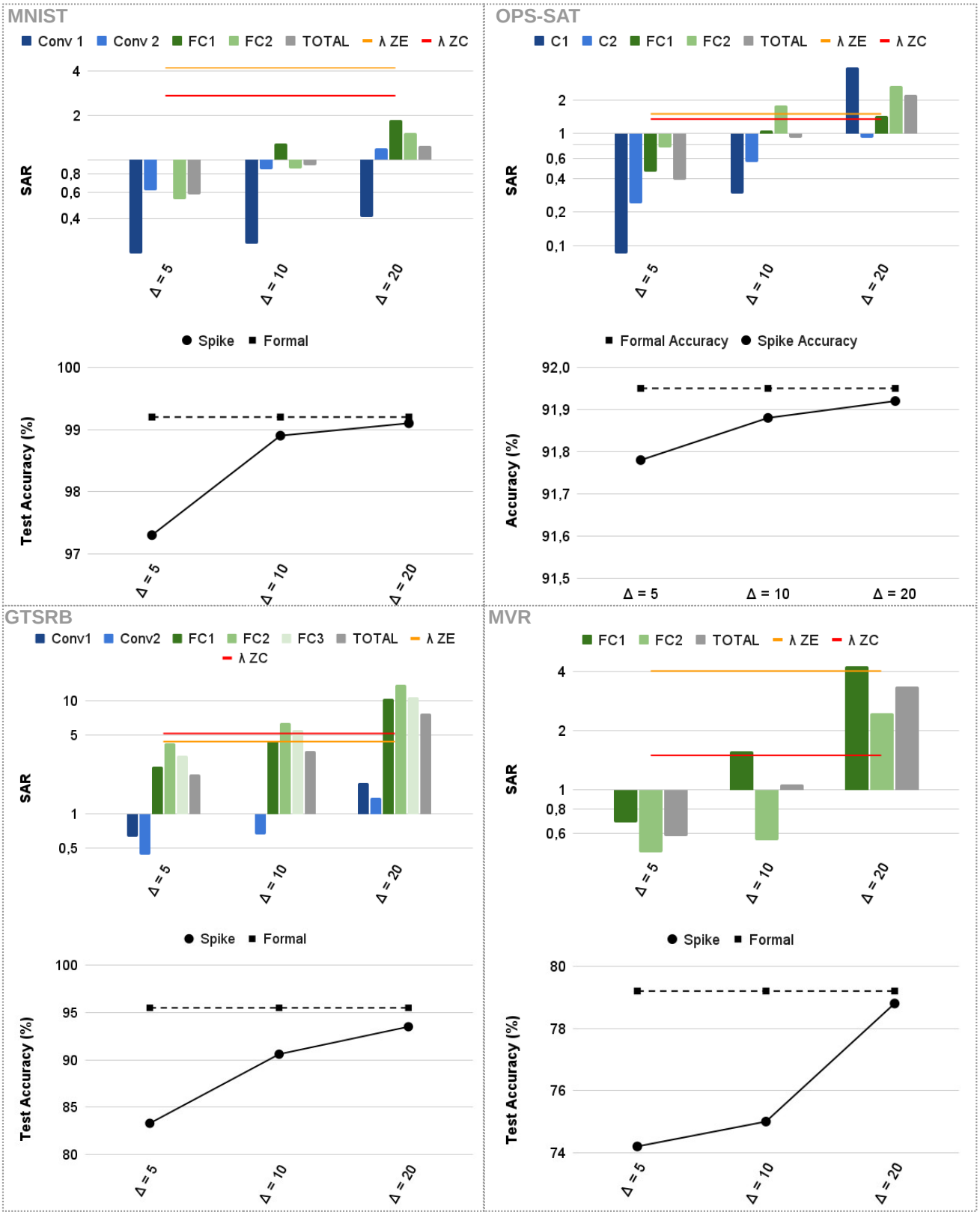


Figure 4.10: SAR (above) and accuracy (below) on OPSSAT, MNIST, GTSRB and Mines Vs Rocks datasets classification, for $\Delta = 5, 10$ and 20. λ_{ZE} and λ_{ZC} are also represented. Spiking inference with N2D2.

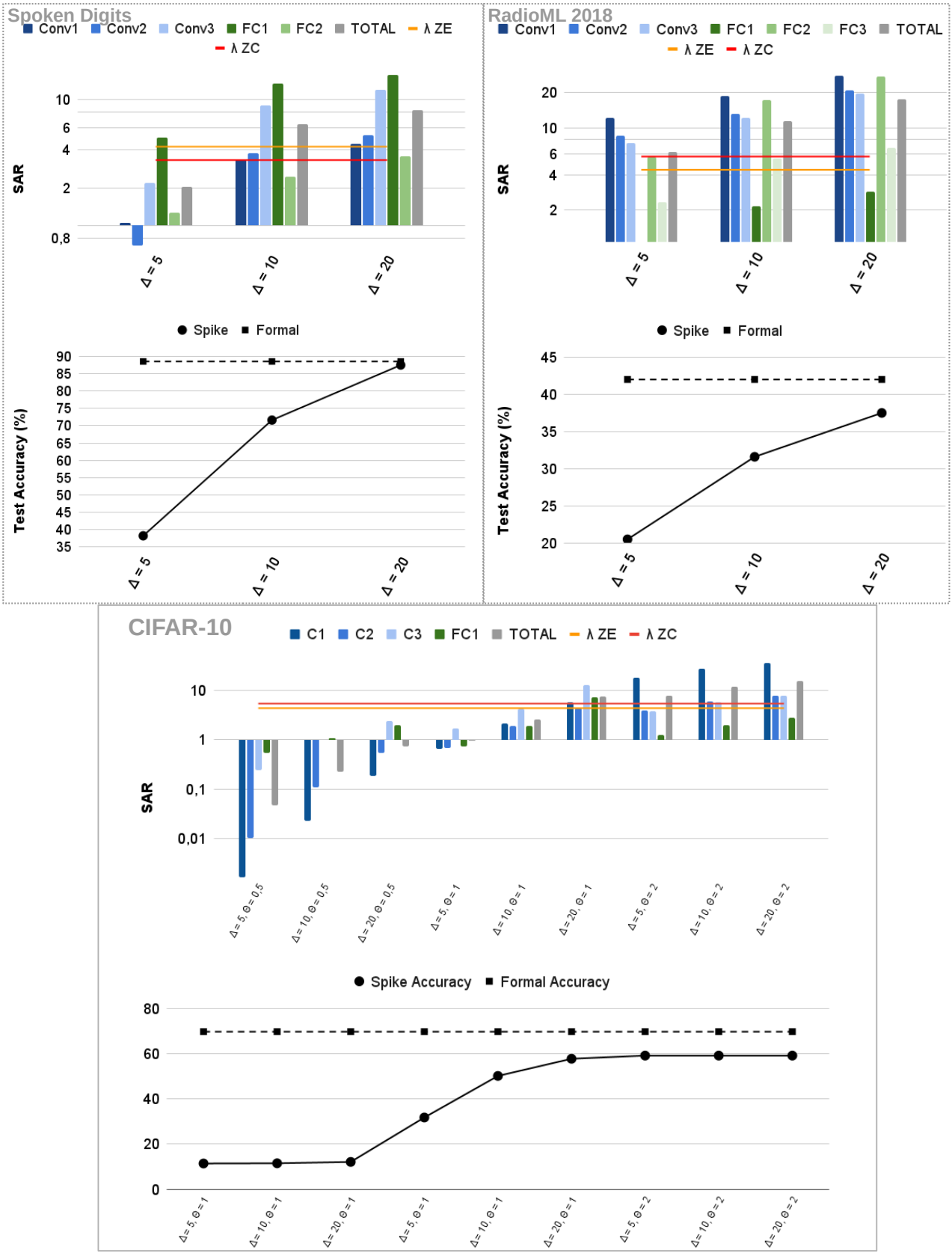


Figure 4.11: SAR (above) and accuracy (below) on Spoken Digits, RadioML 2018 and CIFAR-10 datasets classification, for $\Delta = 5, 10$ and 20. λ_{ZE} and λ_{ZC} are also represented. Spiking inference with N2D2. Additionally, for CIFAR-10 θ_{IF} varies between 0.5, 1 and 2.

Table 4.7: Network-wise Synaptic Activity Ratio for the dataset benchmark, with varying Δ and Θ_{IF} values. The cells are green when SAR is above λ_{ZE} and λ_{ZC}, yellow when it is between the two and red when it is below.

Task	FNN ACC. (%)	Δ	SNN ACC. (%)	SAR
OPSSAT RGB (Std. dev.)	91.95 (0.37)	5	91.78 (0.52)	**0.39**
		10	91.88 (0.83)	**0.92**
		20	91.92 (1.21)	**2.22**
MNIST (Std. dev.)	99.2 (0.2)	5	97.3 (0.6)	**0.58**
		10	98.9 (0.5)	**0.82**
		20	99.1 (0.6)	**1.24**
GTSRB (Std. dev.)	95.5 (0.8)	5	83.3 (5.1)	**2.24**
		10	90.6 (1.8)	**3.59**
		20	93.5 (1.4)	**7.70**
CIFAR-10 (Std. dev.)	69.8 (1.1)	5 $\Theta_{IF} = 0.5$	11.45 (6.1)	**0.05**
		10 $\Theta_{IF} = 0.5$	11.54 (1.9)	**0.23**
		20 $\Theta_{IF} = 0.5$	12.14 (0.9)	**0.73**
		5 $\Theta_{IF} = 1$	31.8 (0.9)	**0.94**
		10 $\Theta_{IF} = 1$	50.2 (2.7)	**2.57**
		20 $\Theta_{IF} = 1$	57.8 (1.9)	**7.57**
		5 $\Theta_{IF} = 2$	59.2 (0.9)	**7.88**
		10 $\Theta_{IF} = 2$	59.2 (0.9)	**11.77**
		20 $\Theta_{IF} = 2$	59.2 (0.9)	**15.64**
Mines VS Rocks (Std. dev.)	79.2 (9.8)	5	74.2 (5.8)	**0.58**
		10	75 (4.4)	**1.07**
		20	78.8 (5.6)	**3.34**
Spoken Digits (Std. dev.)	88.55 (2.3)	5	38.2 (10.1)	**2.04**
		10	71.6 (4.7)	**6.41**
		20	87.5 (4.2)	**8.24**
RadioML (Std. dev.)	42 (0.3)	5	20.5 (5.6)	**6.20**
		10	31.6 (4.9)	**11.45**
		20	37.5 (3.8)	**17.41**

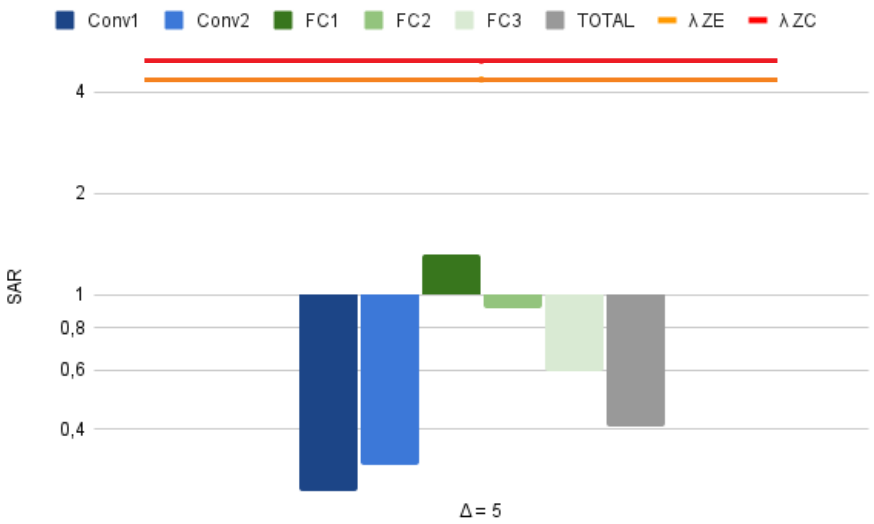


Figure 4.12: Activity ratio on MNIST using a larger topology, obtained using N2D2. Spiking Accuracy = 98,73%; Formal Accuracy = 99,19%

to intrinsic task complexity, and not topology size. In the next section of this chapter (Section 4.4.3), we focus on the relation between data type (spatial or vector) and SAR. The aim is to explain why vector datasets demonstrate higher SAR than images. Our intuition is that vector datasets are not intrinsically unfavorable to spiking domain. Rather, our hypothesis is that the current rate-coding policy (Equation 2.5) is not adapted to the data distribution.

4.4.3 Data type and rate-coding

As explained in section 2.3.2, the rate-coding policy associates spike trains with high frequency to pixels with high intensity, and *vice versa*. The spike train period with respect to input intensity is represented in Figure 4.13. For a better readability, the vertical axis is logarithmic. According to this Figure, rate-coding makes a clear distinction between elements of very low intensity and others. On the other hand, Figure 4.14 provides the distribution of intensity in input data for all datasets of the benchmark. In this figure, image datasets (on the left) features wide intensity distribution, ranging from 0 to 255. On the other hands, the distribution is much narrower for vector datasets (on the right): there is no (or very few) elements of low intensity in such data. The current rate-coding function (Figure 4.13) is therefore not suited for data with narrow intensity distribution. In the current situation, rate encoding implies a drastic loss of information of in the case of vector datasets. This phenomenon explains why such data have poor accuracy and high SARs. The SAR and accuracy could be improved by rescaling input data on $[\![0; 255]\!]$ or by moving to another spike coding strategy.

4.4.4 Layer-wise SAR & hybridization

Besides the aforementioned network-wise results, the SAR was also computed at the layer level following Equation 4.6. In SAR result figures (Figures 4.10 and 4.11), the convolution layers (Conv) are shown in variations of blue, and the fully-connected (FC) layers are shown in nuances

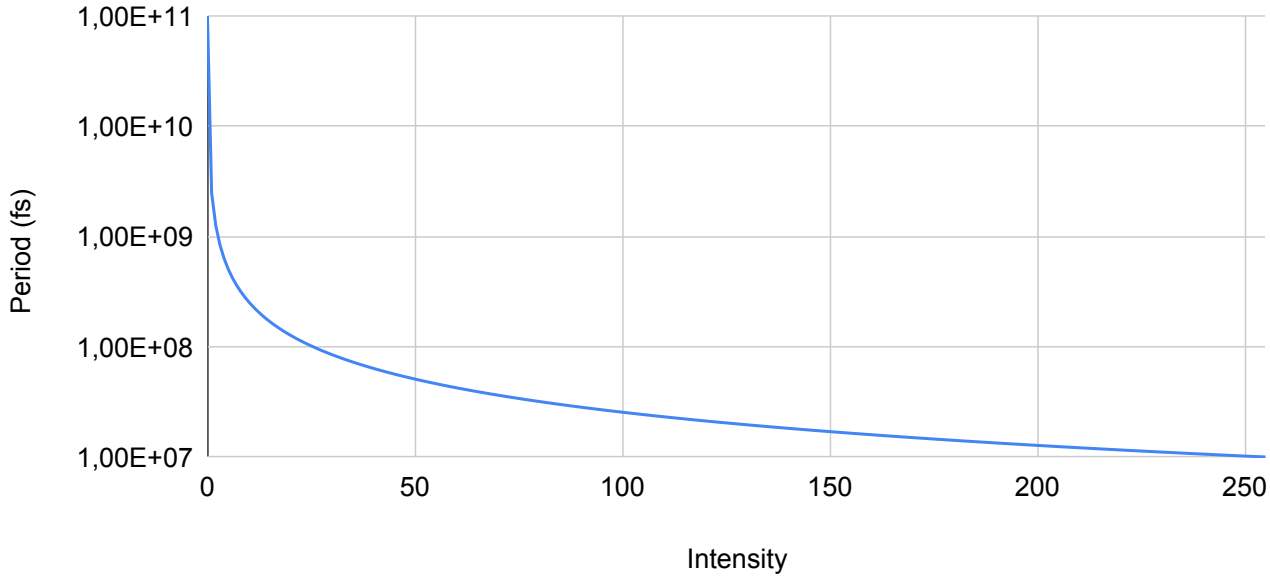


Figure 4.13: Spike train period with respect to element intensity according to the rate-coding policy, with $PeriodMax = 1 \times 10^{11} fs$ and $PeriodMax = 1 \times 10^{7} fs$ (default N2D2 parameters used in our experiments)

of green. This finer granularity level offers a different view on this metric. It shows how SAR varies from one layer to another in the same experiment. As seen in the results, the SAR indeed varies greatly from one layer to the other. For example in RadioML task with $\Delta = 20$, the SAR in layer *Conv1* (24.9) is nine times higher than in *FC1* (2.8). This is the most extreme case of intra-topology SAR variability. The gap is not so high in other experiments, but remains always significant.

According to those layer-wise results and based on the SAR energy model, some layers seem more adapted to spiking domain than others within the same network. Moreover, some layers even show a local SAR favorable to spiking domain (below or between λ_{ZE} and λ_{ZC}), whereas others are not. This is the case in Spoken Digits with $\Delta = 20$; where FC2 is the only layer whose SAR is below λ_{ZE}. In other words, the layer-wise SAR of some layer seem suitable for spiking domain acceleration while the overall SAR does not. Under such circumstances, adapting the coding domain at layer-level could be a fruitful compromise between spiking and formal implementation.

4.5 Conclusion

In this chapter, we proposed a list of machine learning datasets with various levels of difficulty, covering both spatial (images) and vector data (sonar echoes, Fourrier transform of voice recordings and RF signals). This benchmark represents a wide variety of possible applications for embedded neural networks. Synaptic activity was measured on this benchmark using various CNN topologies. The influence of SNN-specific hyper-parameters on synaptic activity and SAR was studied, namely Δ and Θ_{IF}. This study highlighted the trade-off between synaptic activity and accuracy. Moreover, studying the influence of Δ and Θ_{IF} helped us to understand the behavior of spiking neural

networks in various conditions.

We also propose a model to assess the potential energy savings provided by spiking acceleration over conventional formal implementations. This model is based on the Synaptic Activity Ratio metric (*i.e.* average number of spikes per synapse), and on the relative cost of MAC and ACC operations in FPGA. Based on a widespread approximation in literature [78] [31], the model assimilates the energy consumption of an hardware neural network to that of the synaptic operations: MACs in FNNs, ACCs in SNNs. In order to refine this approach, we propose a model to evaluate $\lambda = \frac{E_{MAC}}{E_{ACC}}$ on FPGA targets. The model takes the saturation of DSP into account, which depends on the target device and on the FNN accelerator baseline design (VGT in this case). The position of the SAR relatively to λ therefore defines the suitability of the model to spiking acceleration: $(SAR) < \lambda$ indicates that the spiking model should use less energy than the formal one in hardware. Thanks to the model, we are thus able to propose a fast, high-level cartography of suitable applications for neuromorphic acceleration based on rate-coded SNNs. For instance, the model is applied to a benchmark of datasets using the N2D2 [1] framework.

The results show that simpler tasks provide better SAR independently from topology size. Accordingly, simple tasks seems more suited to rate-based spiking domain. Additionally, spiking domain was more suited to images than 1D data. In response, we propose that tailoring the rate-coding policy to the data dynamic range could be a solution. The same kind of study should be applied to other types of spike coding (time, rank order...). The SAR metric and λ model can be used to determine the suitability of a spike encoding rule to hardware acceleration. Finally, this method could be applied to other families of SNNs: in Chapter 7, we use the SAR metric and λ model on a novel, timestep-constrained SNN model proposed by Neftci *et. al.* [32]. Lastly, we have studied layer-wise SAR, and found great variability of SAR within a same network. Some layers show SAR lower than λ, whereas others don't. More generally, we propose that coding domain hybridization could be an interesting idea, by tailoring the coding domain at a finer granularity, and thus benefiting from spiking domain locally in the architecture. This proposition will be studied further when addressing coding domain hybridization in Chapter 6.

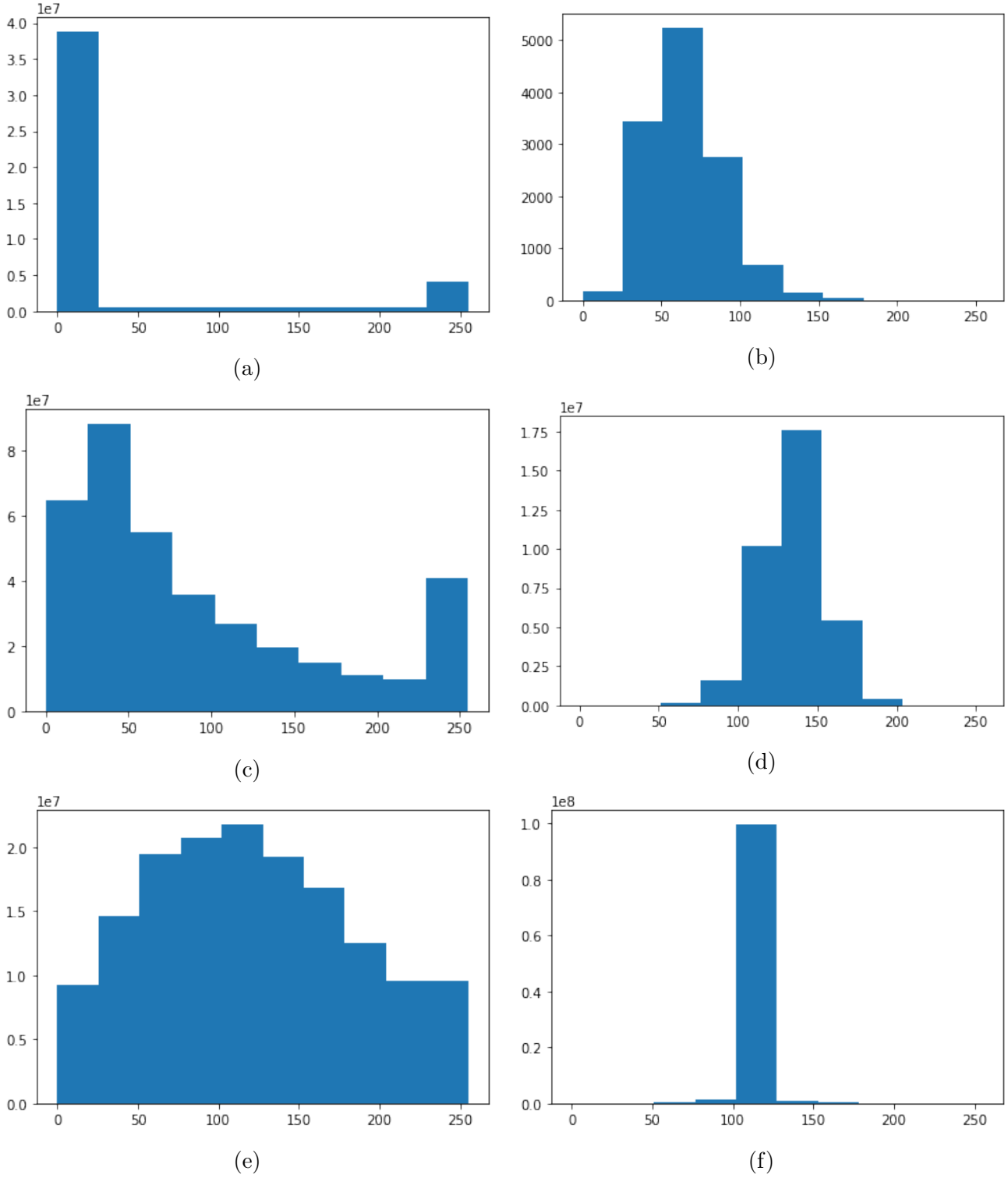


Figure 4.14: Distribution of element intensity in our benchmark datasets. Left are image datasets, right are 1D datasets. a) MNIST, b) Mines VS Rocks, c) GTSRB, d) Spoken Digits, e) CIFAR-10, f) RadioML2018

Chapter 5

Hardware Footprint and High-Level Estimations

Chapter contents

In this chapter, we propose a framework for high level prototyping of neuromorphic accelerators on FPGA. Based on high-level information (description of the CNN and software inference logs), the framework is able to accurately estimate logic resources occupation, power, execution time and energy of various FPGA NN accelerators. The goal of is to enable the exploration of design spaces and application use cases for neuromorphic acceleration. Most importantly, the framework focuses on the comparison of formal and spiking domains under two opposed levels of parallelism. The framework involves estimation for the following designs:

- PADS: the fully-parallel spiking architecture for FC layers introduced in Chapter 3.
- SPLEAT: the fully-sequential spiking architecture described in Section 2.4.2.8.
- VGT: the fully-parallel formal architecture described in Section 2.2.2.1.
- C-HLS: the fully-sequential formal architecture described in Section 2.2.2.2.

The framework is built upon two main bricks: the hardware-footprint database and the spiking inference simulator. The first is a structured set of hardware measurements (LUT,FF, RAM, DSP, Dynamic Power, Execution Time) derived from a vast hardware synthesis and simulation campaign. It should be noted that the database required 264 sets of hardware synthesis, post-synthesis simulation and power analysis using Xilinx Vivado toolchain. That is more than 300 hours of computation, not counting all the software crashes and unused results. Interpolation of the database enables reliable resource estimations. Inference time and dynamic power estimations for formal architectures (VGT and C-HLS) are obtained in the same way, as they only depend on the CNN topology. However in spiking architecture (PADS and SPLEAT), time and power also depends on synaptic activity. That is the role of the second brick of framework: the spiking inference simulator. This software uses information from the hardware footprint database and N2D2 synaptic activity logs. It provides reliable time and power estimations for SPLEAT and PADS. Finally, energy estimations are derived from time and power estimations. The framework is described in details in Section 5.2. The framework is built at layer level. Hence, the database and inference simulator provide layer-wise estimations, which are combined at network-level. A Specific software is developed to simulate network-level pipeline in VGT, PADS and SPLEAT. C-HLS architecture does not feature inter-layer pipeline and is thus not concerned.

The framework is applied to the benchmark of Machine Learning datasets mentioned in Section 4.1. It should be noted that this corresponds to 143 layer-wise measurements. Obtaining hardware measurements would have required 100+ hours of computation using conventional methods. The framework reduces this duration to a few minutes. In doing so, we provide insights and trends on the cartography of applications, coding domain and architectural choices. In this document, explanations focus on OPS-SAT and Spoken Digits datasets for clarity and simplicity purpose, but all the other results are listed in Appendix .2. Resource estimations are provided in Section 5.3, power and inference time in Section 5.4 and energy in Section 5.5. When analyzing the results, the goal will be to find the conditions in which SNNs are preferable over FNNs, and quantify the savings. Moreover, the energy estimations results are used to validate the SAR energy model in Section 5.6.

Additionally, it should be noted that all hardware experiments performed in this section have been made on Xilinx Zedboard. The comparisons between two architectures can be extrapolated to other board, but the absolute values provided are only valid for this specific hardware target. Indeed, low-level optimizations and CMOS technology can change from one board to the other and drastically change the subsequent resource, power and energy estimations.

5.1 Motivations

One of the main goals of this thesis is to provide a cartography of applications, neural coding domains and parallelism. To do so, the most direct approach would have been to provide direct hardware measurements on a defined benchmark. Instead, we have chosen to propose a framework and software for layer-wise estimation. In this section, we provide details on the elements which motivated this approach.

5.1.1 Speed-up cartography and exploration

As mentioned in the introduction, hardware measurements of a single design requires extended periods of time for synthesis, post-synthesis simulation and power analysis. Moreover, the number of parameters involved in our cartography is nearly infinite. For example: the CNN topology, the initial weight distribution, the threshold of Intergate & Fire neurons, the Δ value in the Terminate Delta, the rate encoding frequency ranges... Each combination of parameters requires hours of work and computation, thus it is unrealistic to provide an exhaustive exploration. A framework which enables low-level estimations based on high-level information therefore drastically mitigates the time requires for exploration. In doing so, this work enables to rapidly draw trends in the cartography. The benchmark presented in this section (and in Appendix .2) would have required approximately 500 hours of computation alone. Thanks to the framework, results can be obtained in less than an hour. Moreover, the framework can be used by other students and researchers to extend the benchmark proposed in this Chapter.

5.1.2 Layer-wise approach

The layer-wise approach of the framework is motivated by three distinct aspects. First, layers are the basic bloc of feed-forward neural network topologies. Providing layer-wise results for the most common types (convolution, max-pooling and fully-connected) enables to reconstruct a wide range of CNN topologies. Second, fully-parallel implementations of large designs such as RadioML 2018 CNN (Table 4.2) often overpass the available resources of the target board (Zedboard). In such conditions, synthesis duration is largely extended. Moreover, the results are distorted by low-level optimization, altering generalization of measurements. Performing synthesis at layer-level solves this issue as designs are smaller. Third and last, the layer-wise approach gives the opportunity to study the possibilities of hybridization. This feature will be used further in Chapter 6.

5.1.3 Level of parallelism

The four architectures that compose our benchmark are either fully sequential or fully parallel. Those are only prototypes and a realistic architecture should involve an intermediate level of parallelism. However, studying the two extremes of the spectrum yields insights on the underlying trends. More than proposing a real benchmark for the four chosen architecture, the goal is rather to study the influence of parallelism on the comparison between formal and spiking coding domains. Moreover, the layer-wise estimations will also provide insights on the tuning of parallelism at layer level in spiking architectures. Such information lacks in literature and could facilitate the adoption of spiking technology in industry.

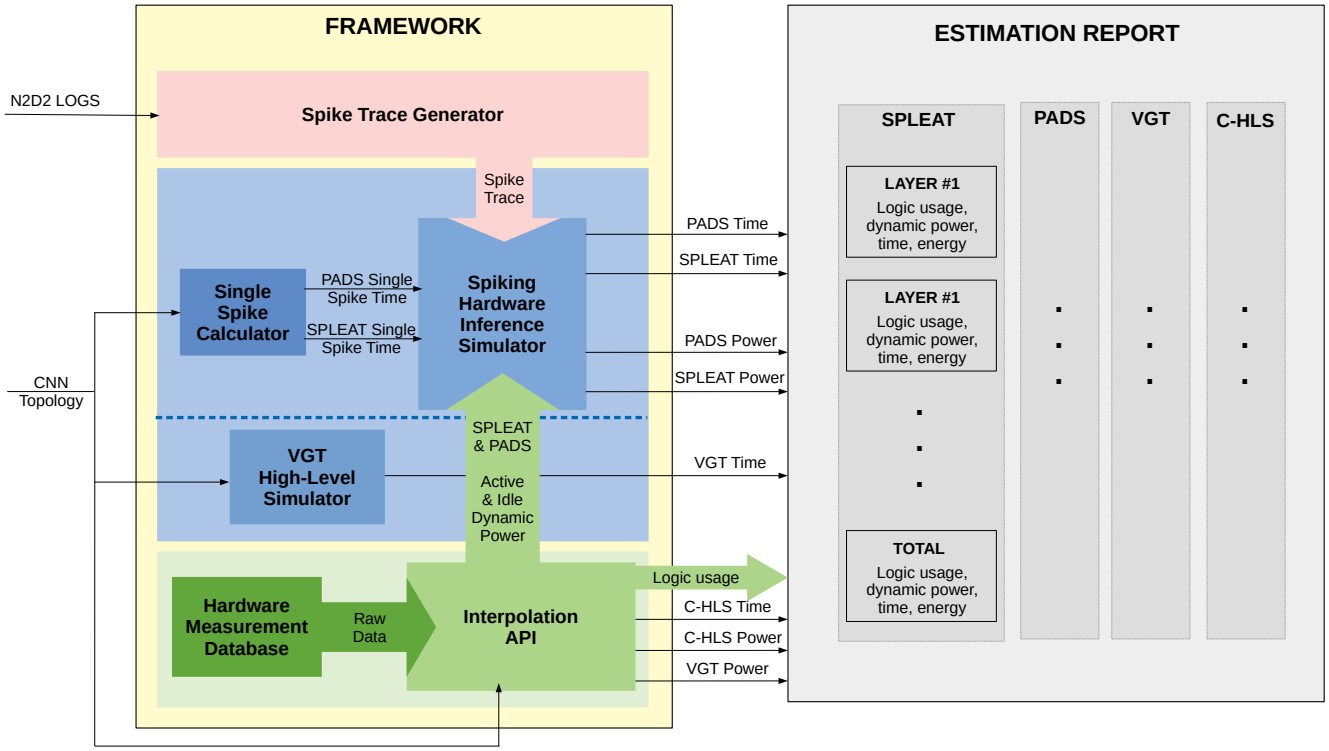


Figure 5.1: Overview of the high-level estimation framework

5.2 Framework

In this section, we describe the high-level estimation framework in details. An overview of the framework is available in Figure 5.1. The structure is divided in three main areas, illustrated in shades of red, green and blue. Each area corresponds with one section of the framework. The green section addresses the hardware measurement database. It also involves the python API developed for interpolation of measurements. The logic resources estimations (LUT, FF, RAM, DSP) for all architectures are obtained directly after interpolation, as well as VGT dynamic power and C-HLS power and execution time. The green area will be described in Subsection 5.2.1. The red section addresses the generation of spike traces from the N2D2 activity logs. It has already been described in the previous chapter (Section 4.4.1). Finally, the blue section covers the inference simulators. The role of this area is to compute dynamic power and inference times for spiking accelerators based on spike traces, active and idle power from the database, and single spike execution times. The single spike execution times are computed separately (Subsection 5.2.2.3). The hardware spiking inference simulator is described in Subsection 5.2.3. Additionally, a dedicated high-level behavioral simulator computes VGT inference times (Subsection 5.2.2.2). It has been developed to simulate the inter-layer pipeline in this architecture.

5.2.1 Hardware measurements database

We begin by describing the green area of the framework in Figure 5.1. That is, the hardware measurement database and python interpolation API. In a first time, we describe the structure of the database *i.e. design spaces* and the experimental protocol settled for hardware measurements.

Then, we describe and shortly discuss the 3D interpolation of hardware metrics on the design spaces.

5.2.1.1 Design Spaces & Measurement Campaign

The database should provide information at layer level for each of the four architectures (SPLEAT, C-HLS, PADS and VGT). There are 3 types of layers: Convolution (CONV), Pooling (POOL) and Fully-Connected (FC). The database is thus divided in 12 sets, one for each combination of layer and architecture. Those sets are called "design spaces" in the remaining of this work. We focus on 2D design spaces, *i.e.* design spaces that are governed by two parameters. The number of dimensions of a design space corresponds to the number of considered variables (*i.e.* hyper-parameters). In our case, the study is limited to 2D design spaces. The corresponding hyper parameters are:

- For Convolution layers: Number of filters (N_{fil}) and width of input (S_{in});
- For Pooling layers: Number of input channels ($N^{\text{in}}_{\text{chan}}$) and input width ($N^{\text{out}}_{\text{chan}}$);
- For Fully-Connected layers: Number of input neurons ($N^{\text{in}}_{\text{neur}}$) and number of output neurons ($N^{\text{out}}_{\text{neur}}$).

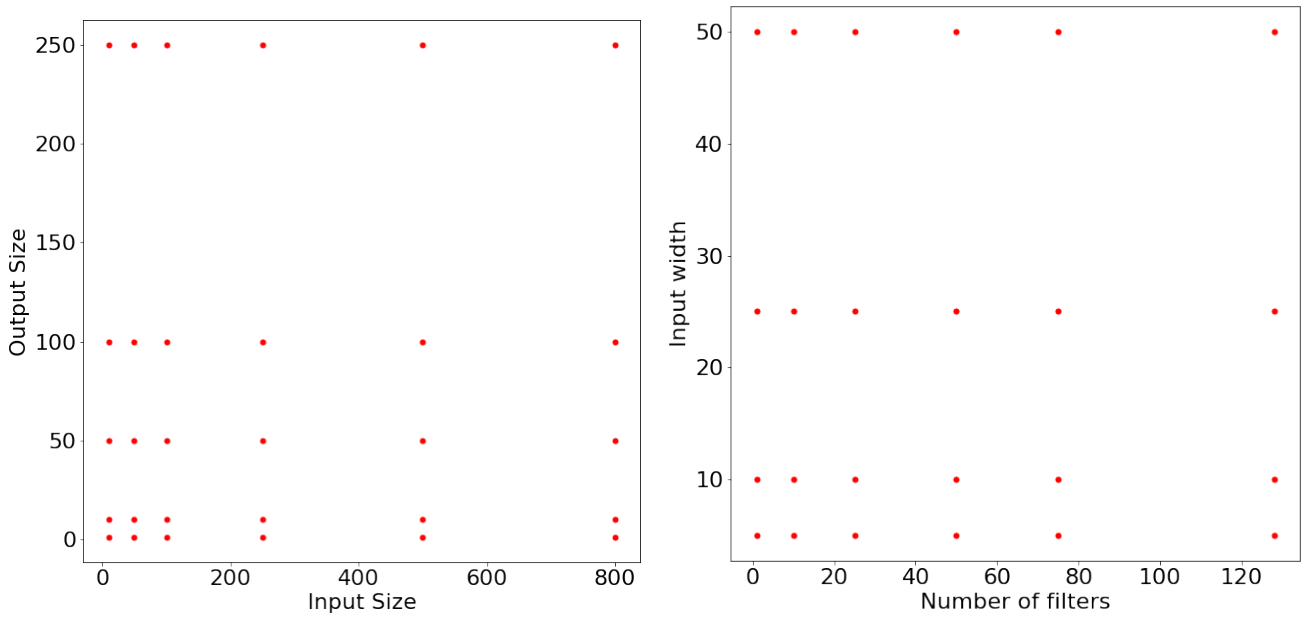
There are other hyper-parameters such as the number of input channels in convolutions or the stride in max-poolings. However, each additional dimension multiplies the number of hardware synthesis required to build the database. Using more than 2D design spaces would imply an unmanageable amount of time. A preliminary study (which is not detailed here) have shown that hardware footprint is mostly governed by the two selected hyper-parameters. That is, apart from the number of input channels in convolution layers ($N^{\text{in}}_{\text{chan}}$). This parameter has as strong influence that must be taken in account. In a dedicated paragraph at the end of this subsection, we provide the approximation made to acknowledge this parameter. However, $N^{\text{in}}_{\text{chan}}$ is not considered in the convolution design space, and the approximation is only applied during interpolation.

Each design space is thus defined by the range of its defining variables. For each layer, the ranges are tuned to cover all layers involved in the CNNs of the benchmark benchmark (Section 4.1.1). It should be noted that the first fully-connected layer (FC1) of RadioML 2018 CNN is not covered, due to its eccentricity. RadioML-FC1 has 15872 input neurons, where the second largest layer only has 800. It is thus omitted in all the following results.

- For Convolution layers: N_{fil} varies from 1 to 50, and S_{in} from 1 to 128 (128×128p images);
- For Pooling layers: N_{fil} varies from 1 to 50, and S_{in} from 1 to 128;
- For Fully-Connected layers: $N^{\text{in}}_{\text{neur}}$ varies from 1 to 800 and $N^{\text{out}}_{\text{neur}}$ varies from 1 to 250.

The other parameters have the following default values:

- For Convolution layers: Filter size is 5×5, Stride is 1 and $N^{\text{in}}_{\text{chan}} = 1$ during the measurement campaign;
- For Pooling layers: Filter size is 2×2, Stride is 2 and $N^{\text{in}}_{\text{chan}} = N_{\text{filters}}$;
- For Fully-Connected layers, there are no other hyper-parameters.



(a) (b)

Figure 5.2: Design space for a) FC layers (30 points), b) Conv & Pool layers (24 points each)

We select a set of points in each design space. 30 for Fully-connected layers, and 24 for Convolution and Pooling. Those points are the same for each architecture, except for PADS which does not cover Convolution and Pooling. At each point, we measure the hardware footprint of each accelerator: PADS, VGT, SPLEAT and C-HLS. The resulting Design Spaces with measurement points are shown in Figure 5.2a for FC layers and Figure 5.2b for convolution and pooling layers. The design spaces of convolution and pooling layers are identical.

The synthesis and measurement protocol for each point is the same as described in Section 3.2.1. For a given measurement point, each architecture is configured according to the coordinates of the point. The four designs are synthesized independently using Vivado Design Suite. A VHDL testbench is configured for each architecture and used for post-synthesis simulation using Vivado XSIM. This VHDL testbench contains realistic input stimuli for the layer. Those stimuli are inspired from previous experiments. During post-synthesis simulation, all signal switches are logged in a SAIF file (Switching Activity Interchange Format). This file is then used in the Vivado Power Analyzer to estimate dynamic power consumption. The realistic stimuli in the testbenches ensure an accurate estimation. Finally, logic resource and power reports are stored in XML format. This protocol is repeated four times (four architectures) for each of the 78 measurement points. All the synthesis, post-synthesis simulation, reporting and storage has been automated via bash and TCL scripts. A bash scripts iterates over all measurement points, and launches a dedicated TCL script for each. The TCL script opens Vivado, loads source files, performs synthesis and post-synthesis simulation. Afterward, a TCL command automatically generates and stores power and resources reports in XML format.

5.2.1.1.1 $N^{\text{in}}_{\text{chan}}$ in convolution layers As explained in previous section, $N^{\text{in}}_{\text{chan}}$ is not considered in the design space of convolution layers. In this section, we explain how it is approximated differently for each architecture.

- PADS is not concerned at it does not feature convolution.

- SPLEAT is event-based, thus the only change is the number of bits required to encode an event (the channel address is encoded in binary). Preliminary experiments have shown that the impact of such modification is negligible even for large number of input channels.
- C-HLS is fully-sequential, thus the number of input channels multiplies the execution time but does not affected resources and power. The number of channels is therefore addressed later
- In VGT, input channels are processed in parallel. Thus, input channels multiplies the logic resources and power footprint, but doesn't affect execution time.

5.2.1.1.2 Idle power usage measurements PADS and SPLEAT are spiking accelerators which use a sparse temporal information coding. That is, the circuit is in idle state between two spikes. Consequently, the overall dynamic power usage is computed as a weighted average of active and idle power consumptions in the spiking hardware inference simulator (Subsection 5.2.3.2). Hence we measure both active and idle power consumption for spiking architecture. That is, using two distinct testbenches: one with regular input stimuli, and the other with empty signals.

5.2.1.2 3D interpolation

All the raw results of the aforementioned measurement campaign are available in tables in Appendix .2. As explained previously, those results are used in estimation through interpolation. For example, in order to obtain the LUT usage for a PADS FC layer with a given configuration, all LUT measurements for PADS FC design space are placed on a regular grid and interpolated using the python Scipy Interpolate library. This mechanism is fully automated through a python software which retrieves the data in the XML files, performs interpolation and retrieves the result. The logic resource estimations for all architectures, as well as dynamic power for VGT and C-HLS are directly obtained through interpolation. For SPLEAT and PADS, inference simulation is required to obtain accurate estimations based on interpolated active and dynamic power. In this subsection, we show the 3D surface graphs resulting for interpolation of all measurements. Those results provide interesting preliminary insights on the design space exploration. For that purpose, they are presented in three sets of comparisons:

- A comparison between SPLEAT and C-HLS, to visualize the influence of spiking domain on sequential architectures
- A comparison between PADS and VGT, for the same influence on parallel architectures
- A comparison between SPLEAT and PADS, for the influence of parallelism on spiking domain in hardware

In this section, we limit our scope to Fully-Connected layers. First because PADS only covers FC layers, and second because the results are similar for convolution and pooling layers in other architectures. Moreover, the number of graphs and discussions would be two great covering all three types of layers.

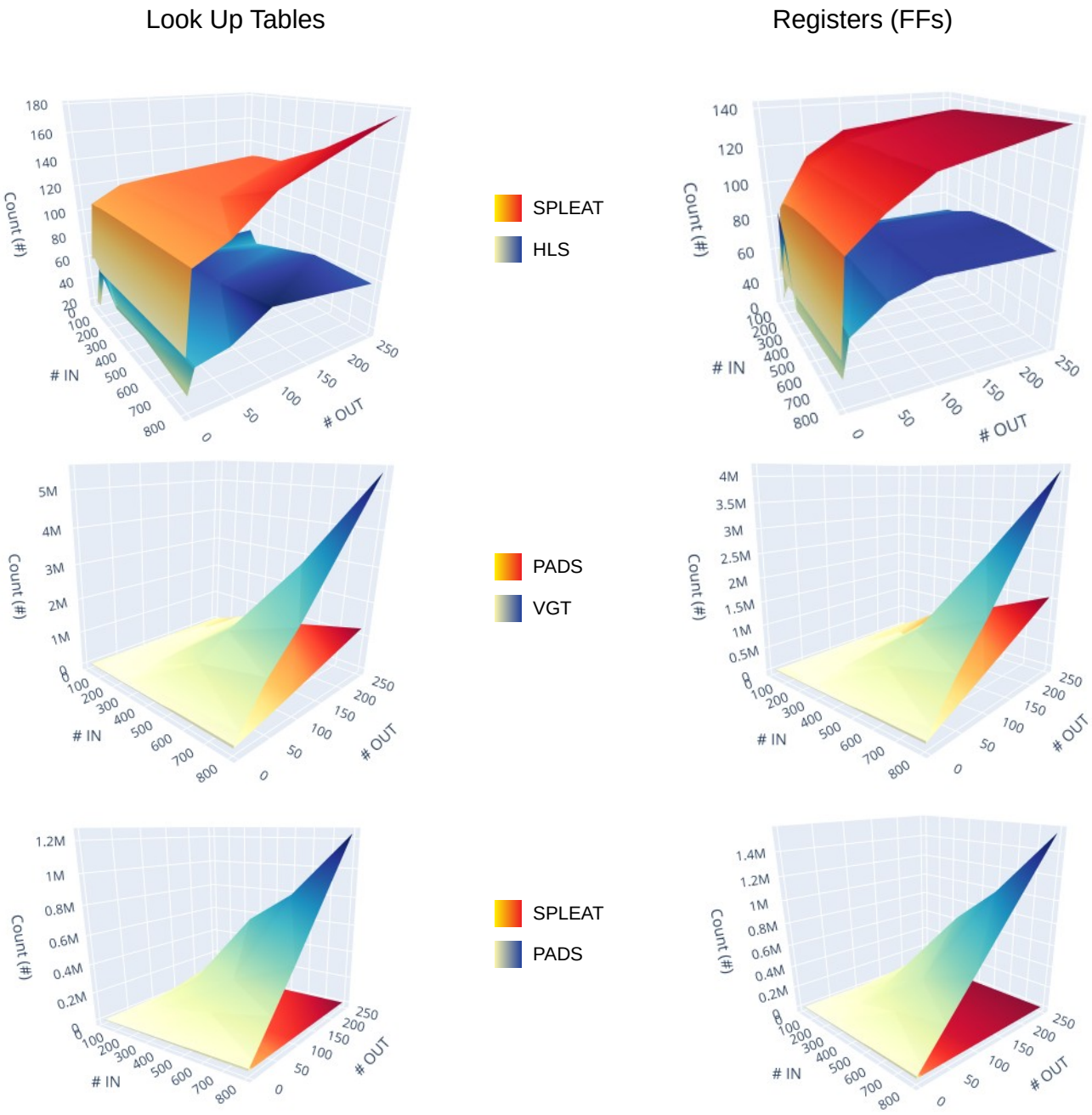


Figure 5.3: 3D interpolations of LUTs (left) and FFs (right) for Fully-Connected layers. First row: SPLEAT (red) vs HLS (blue), second row: PADS (red) vs VGT (blue). Third row: SPLEAT (red) vs PADS (blue).

5.2.1.2.1 Look-Up-Tables & Registers The results for LUTs and Registers (Flip-Flops, FF) are shown in Figure 5.3. LUTs and FFs are the two basic logic elements of the Programmable Logic. Every design uses some, which is not the case of more specialized elements like RAM or DSP. Moreover, RAM and DSPs can be synthesized in LUTs and FFs when the dedicated elements are saturated. Thus, LUTs and Registers usage is a global indicator of FPGA occupation. The interpolation surfaces for LUTs (left) and FFs (right) are shown as 3D surfaces. The first row depicts SPLEAT against C-HLS, the second depicts PADS against VGT, and the third SPLEAT vs PADS.

5.2.1.2.2 Memory & DSPs The 3D surface plots for Block RAM (BRAM) and DSP usage in Fully-Connected layers are provided in Figure 5.5. RAM and DSPs are specialized logic elements, in contrast with LUTs and Flip Flops. DSP are used for multiplication-accumulation operations. Block Memory is used to store data mostly in sequential architectures. As explained in previous subsection, LUTs and FFs can be used instead of RAM and DSP when such scarce resources are saturated. This could affect logic occupation and power usage. Thus, those two resources are also valuable indicators of FPGA occupation.

5.2.1.2.3 Power The power interpolation results are divided in two metrics: active and idle power usages. Active power corresponds to a situation where the architecture is receiving and processing input stimuli. On the other hand, idle power corresponds to an architecture in idle state. Active power is measured for all architectures, but idle power is only considered in spiking architectures. Indeed, only spiking architecture are susceptible of being in idle state during some part of processing. The results of interpolation on active power measurement are given in Figure 5.6. The interpolation of idle power on fully-connected spiking layers are given in 5.4. In idle power figures, active power is also represented for comparison.

The results shows that idle power is very similar to active power in all cases. Thus, the design does not benefit from the event-based coding in terms of energy consumption. Power-gating or clock-gating techniques might be used to nullify power usage in idle state. This possibility will be explored further in outlooks and discussions. The results presented in this section give a global view resource and power usages in neuromorphic accelerators depending on parallelism and coding domain. If sequential accelerators might seem more scalable at first, this is not guaranteed for larger topologies because of RAM saturation. However, SPLEAT demonstrates drastically lower resource and power usages than PADS on current design spaces. On the other hand, SPLEAT uses a little more resources and power than C-HLS, although the difference is not significant in front of the available resources. PADS demonstrate both lower power and resource usage than VGT across all design space. Moreover, the metrics increases faster for VGT. Thus, PADS is more scalable than VGT among parallel implementation both in terms of logic resources and power usage. Finally, idle power in spiking architecture is very close to active power. Thus, no major power savings should be expected from the asynchronous behavior of PADS and SPLEAT.

5.2.2 Execution time

In this section, we describe how are made estimations of inference time. The method is different for each architecture.

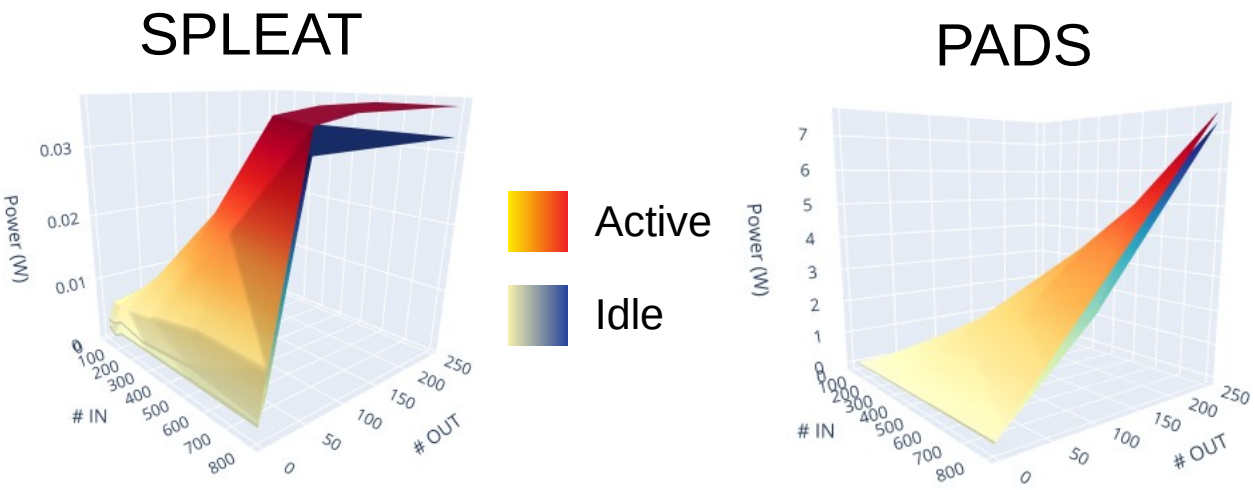


Figure 5.4: 3D interpolations of active and idle power for Fully-Connected layers. Left: SPLEAT, right: PADS.

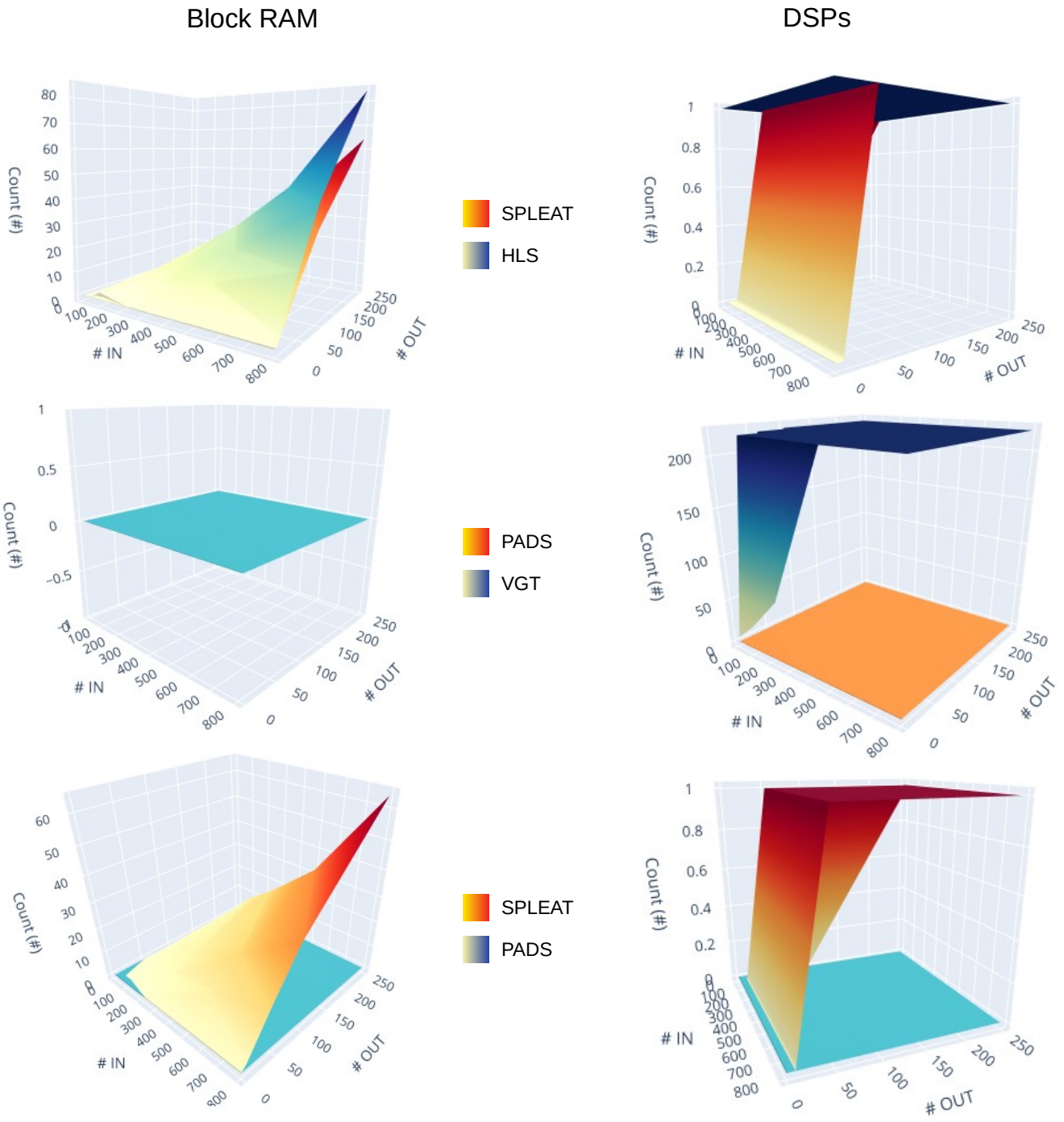


Figure 5.5: 3D interpolations of Block RAM (left) and DSPs (right) for Fully-Connected layers. First row: SPLEAT (red) vs HLS (blue), second row: PADS (red) vs VGT (blue). Third row: SPLEAT (red) vs PADS (blue).

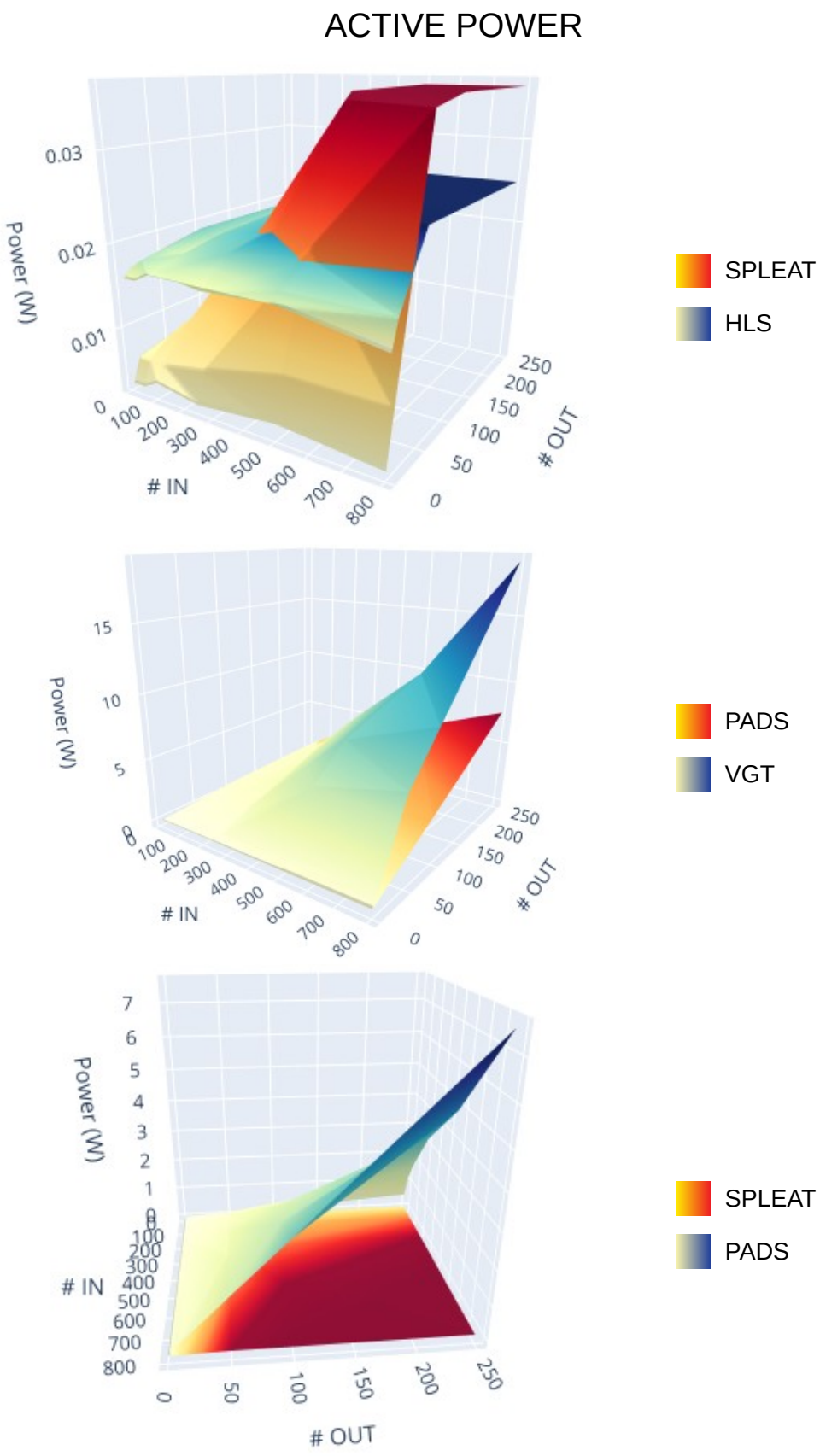


Figure 5.6: 3D interpolations of Active Power for Fully-Connected layers. First row: SPLEAT (red) vs HLS (blue), second row: PADS (red) vs VGT (blue). Third row: SPLEAT (red) vs PADS (blue).

5.2.2.1 C-HLS inference time interpolation

C-HLS is fully sequential and does not feature inter-layer pipeline. That is, the exeuction time of a layer only depends on its configuration. Hence, execution time in C-HLS is dealt with as resource estimations: it is measured at each measurement point in the design spaces, and interpolated using the same *Scipy Interpolate* python library. Moreover, the total execution time is the sum of layer-wise results, as no pipeline is involved.

5.2.2.2 VGT behavioral simulation

VGT features inter-layer pipeline, therefore the behavior of previous layers affects the following ones. Consequently, the inference time of a layer does not only depend on the configuration of the layer, but on the configuration of all previous layers. A simple python software is thus developed for high-level behavioral simulation of VGT. The simulator reproduces the behavior of VGT layers and inter-layer pipeline without performing the actual computations. It has been developed by analyzing the VHDL code and studying behavioral simulations of the architecture in Modelsim software. Moreover, the simulator has been validated on several cases by comparison with Modelsim behavioral simulations. It is used to provide accurate layer-wise and network inference time estimations for VGT.

5.2.2.3 Spiking architectures: single spike execution time

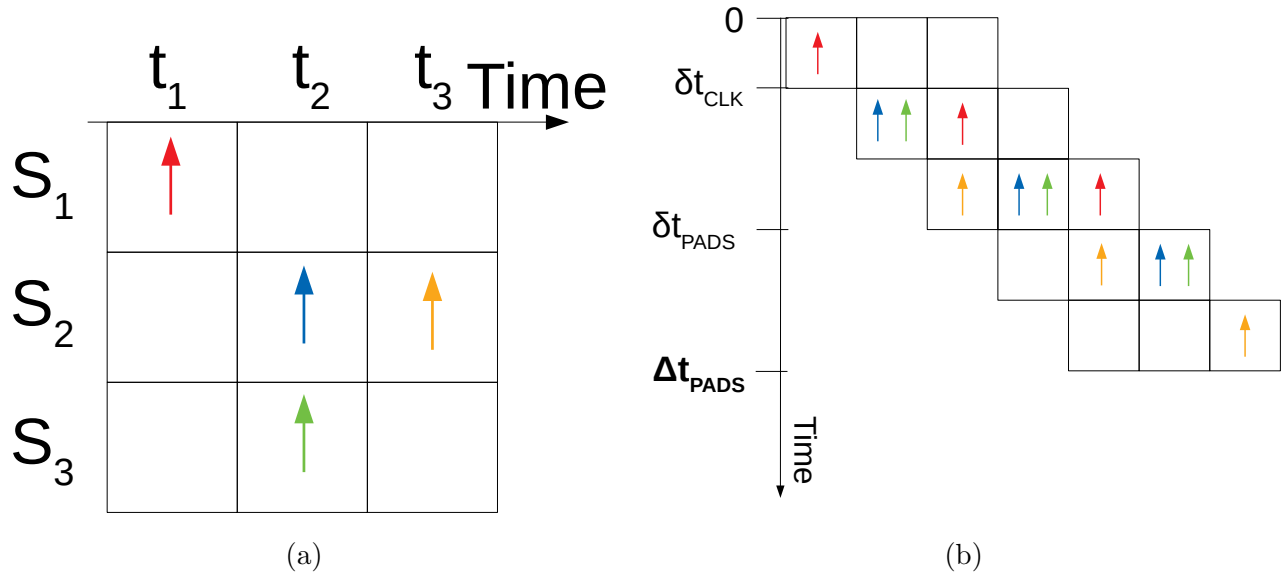
In spiking architectures, inference time is estimated using a dedicated spiking inference simulator. The simulator is based on two elements: the synaptic traces obtained from N2D2 spiking inference, and the single spike execution time. The simulator will be detailed later in Subsection 5.2.3. In this paragraph, we provide information on those single spike execution times. That is, the time required to process a single input spike in either SPLEAT or PADS. In order to obtain equations that define execution time with respect to layer configuration, the VHDL code of SPLEAT and PADS is analyzed in depth. The analysis yields to Equations 5.1 and 5.2 for execution times (in clock cycles) of SPLEAT and PADS respectively.

$$\begin{aligned} \delta t_{\text{SPLEAT Conv}} &= 4 + (\frac{K}{S})^2 \times N_K \\ \delta t_{\text{SPLEAT Pool}} &= 1 + K^2 \\ \delta t_{\text{SPLEAT FC}} &= 3 + N_{\text{out}} \end{aligned} \tag{5.1}$$

With δt the processing times (in clock cycles) for a single spike, K the kernel size (five for convolutions, two for max-pooling), S the stride (one for convolutions, two for max-pooling), N_K the number of kernels, and N_{out} the number of output neurons. For a better understanding of those equations, more material on SPLEAT is available in [14].

$$\delta t_{\text{PADS FC}} = 2 + log_2(N_{\text{input}}) \tag{5.2}$$

With δt the processing time (in clock cycle) for a single spike and N_{input} the number of input neurons. This equation directly illustrates the process of the Generic Neural Processing Unit (NPU) of PADS shown in Figure 3.4 (Section 3.1.2). Indeed, $log_2(N_{\text{input}})$ is the depth of the adder tree in the Generic NPU. The 2 additional clock cycles are caused by the threshold & firing process.



(a) (b)

Figure 5.7: a) Simple spike trace example with 4 input synapses and 4 time increments, b) Illustration of PADS parallel and pipelined process.

The two equations are verified on several measurement points using Modelsim behavioral simulation before the measurement campaign. Both equations are integrated in python functions to be used later in the spiking hardware inference simulator (Section 5.2.3).

5.2.3 Spiking hardware inference simulator

In this subsection, we describe the spiking hardware inference simulator used to estimate inference time and dynamic power usage of spiking architectures. It is based on spike traces derived from N2D2 synaptic logs, single spike processing times, as well as active and idle power interpolations.

5.2.3.1 Inference time

Inference time estimation in SPLEAT and PADS is based on the same spike trace used by the SAR estimator in Section 4.4.1. The spike trace is converted from N2D2 format to a unified format. That is, each layer is associated with an spike trace in the form of a 2D matrix of size N×M. N is the number of input synapses, and M the number of time increments. Indeed, time is discretized in increments of one clock cycle (10ns at 100MHz). As a reminder, N2D2 spiking inference is configured to mimic the spike generation process of the GenCell. Thus N2D2 synaptic activity effectively represents the real input stimuli of hardware spiking layers, as it involves the exact same network with identical hyper-parameters. A simple spike trace with 4 output synapses and 4 time increments (*i.e.* 4 clock cycles) is provided in Figure 5.7a. The inference time estimation is also based on the single spike processing times (δt_{PADS}) and δt_{SPLEAT} obtained in 5.2.2. The estimation protocol is different for PADS and SPLEAT, as described below.

5.2.3.1.1 Simulation for PADS PADS is a fully-parallel pipelined architecture, therefore it is able to receive up to one spike per synapse and per clock cycle. In other words, PADS is able to process one spike trace column per clock cycle. The pipeline process is illustrated in Figure 5.7b. In this Figure, $\delta_{t\text{PADS}} = 3$ clock cycles, *i.e.* the pipeline is three stage deep. A python simulator

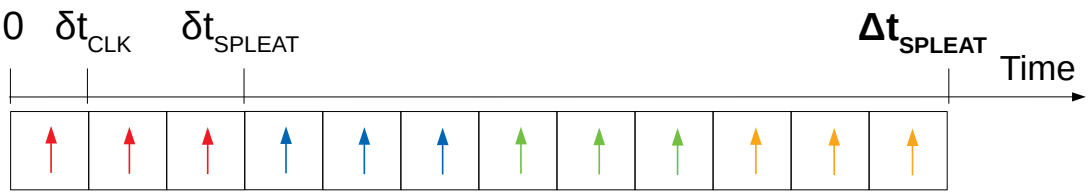


Figure 5.8: Illustration of SPLEAT sequential process.

is developed to compute execution time based on spike traces derived drom N2D2. Knowing the values of $\delta_{t\text{CLK}}$ (10ns at 100MHz) and $\delta_{t\text{PADS}}$ (Section 5.2.2), it is able to provide accurate time estimations from high-level spike trace information.

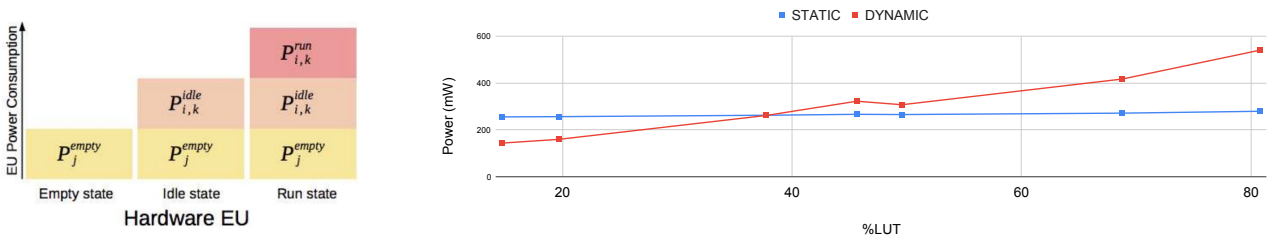
5.2.3.1.2 Simulation for SPLEAT SPLEAT on the other hand is a highly sequential, with no intra-layer pipeline. Hence, spikes must be processed one by one. SPLEAT is thus able to process at most one spike-trace cell per clock cycle. SPLEAT is event-based, thus empty cells are not processed. The ensuing process is illustrated in Figure 5.8. Like for PADS, a python simulator is developed to compute inference time based on spike traces, δ_tCLK and $\delta_{t\text{SPLEAT}}$ values. In the figure, $\delta_{t\text{SPLEAT}} = 3$, like in PADS figure

Finally, SPLEAT and PADS are both event-based architectures which feature inter-layer pipeline. The propagation of spikes in the network is neglected in front of the actual processing time, thus we consider that all layers are active simultaneously. In consequence the network-level execution time is computed as the maximum layer-wise execution time in both PADS and SPLEAT. In other words, the network-wise inference time is the maximum layer-wise inference time (among all layers of the architecture).

5.2.3.2 Power estimator

In FPGA, power consumption is divided between static and dynamic components. Dynamic power consumption ($P^{\text{Dynamic}} = P^{\text{run}}$) is the power consumed by signal switches inherent to the processing state. Static power consumption is the power consumed by the designed when powered on, but without any clock or signal input. Static power consumption is composed of the board static consumption (P^{empty}), and the design static consumption (P^{idle}). This FPGA energy model was proposed in [105] and illustrated in Figure 5.9a. Moreover, let $P^{\text{Static}} = P^{\text{empty}} + P^{\text{idle}}$. Experiments show that P^{idle} is negligible in front of P^{empty}, thus we consider $P^{\text{Static}} = P^{\text{empty}}$. The results of such experiment is illustrated in Figure 5.9b. It shows evolution of dynamic and static power consumption against FPGA occupation. Static power consumption is not affected by design size, thus $P^{\text{Static}} = P^{\text{empty}}$. Thus, in the model P^{Static} is considered as an offset which depends on the device. For the Zedboard that is 0.250W.

For formal architectures (VGT and C-HLS), dynamic power consumption depends only on configuration. It is thus directly obtained through interpolation of the power measurements database. For spiking architectures on the other hand (PADS and SPLEAT), power consumption depends on active power consumption, idle power consumption and synaptic activity. As a reminder, active power consumption corresponds to the Neural Processing Unit (NPU) state when it is busy processing spikes. Idle power corresponds to the NPU state when awaiting input stimuli. Both have been measured for SPLEAT and PADS in Section 5.2.1.2.3. Dynamic power estimation is computed as an average between active and idle state. The python code for inference time esti-



(a) (b)

Figure 5.9: a) FPGA power consumption model, b) Dynamic and Static power VS LUT occupation (%) on Zedboard

mation is reused to count the number of active and idle clock cycles during processing. Like for inference time, PADS and SPLEAT dynamic power consumption is estimated separately. Based on an input spike trace, the simulator outputs the proportion of time passed in active (n_{active}) and idle (n_{idle}) states, in number of clock cycles. Dynamic power estimation is computed according to Equation 5.3. It should be noted that $n_{\text{active}} + n_{\text{idle}}$ is equal to Δt_{PADS} or Δt_{SPLEAT} for PADS or SPLEAT respectively.

$$P^{\text{Dynamic}} = \frac{n_{\text{active}} \times P^{\text{active}} + n_{\text{idle}} \times P^{\text{idle}}}{n_{\text{active}} + n_{\text{idle}}} \tag{5.3}$$

5.3 Ressource estimations

The estimation framework is applied to the benchmark of seven datasets: MNIST, OPS-SAT, GTSRB, CIFAR-10, Mines VS Rocks, Spoken Digits and RadioML 2018. All raw estimation results are available in Appendix .2. In this document, we focus on two interesting cases: OPS-SAT and Spoken Digits tasks. This section is dedicated to the logic resource estimations of the four accelerators (VGT, PADS, C-HLS and SPLEAT) on the benchmark of datasets. That is Look-Up-Tables (LUTs), Registers (Flip-Flops, FFs), Block RAMs (RAMs) and Digital Signal Processors (DSPs) estimations. Before going into further details, a few notation should be introduced to simplify explanations. The estimations are noted $Metric_{Arch}^{Obj}$, where:

- *Metric* is the considered resource (LUT,FF,DSP,RAM),
- *Obj* is the considered object, it can be a full CNN (MNIST, OPS-SAT...), a sub-part of the CNN (CL: classification, FE: feature extraction) or a specific layer (C1: first convolution, P2: second pooling, FC1: first fully-connected),
- *Arch* is the considered architecture (PADS,VGT,SPLEAT,C-HLS).

Therefore, LUT_{VGT}^{MNIST} refers to the LUT usage the full MNIST CNN implemented with VGT. $RAM_{PADS}^{GTSRB\text{-}FC2}$ refers to the RAM usage of the second FC layer of GTSRB implemented with PADS. Consequently, the ratio between two estimations is noted $\frac{Metric_{Arch}^{Obj}}{Metric_{Arch}^{Obj}}$. It is used to describe savings or overheads between two architectures.

Moreover, the estimation results discussed through three sets of comparisons: PADS vs. VGT, SPLEAT vs. C-HLS and SPLEAT vs. PADS. The two main goals of the study are indeed the comparison of formal and spiking domains and the influence of parallelism on spiking accelerators.

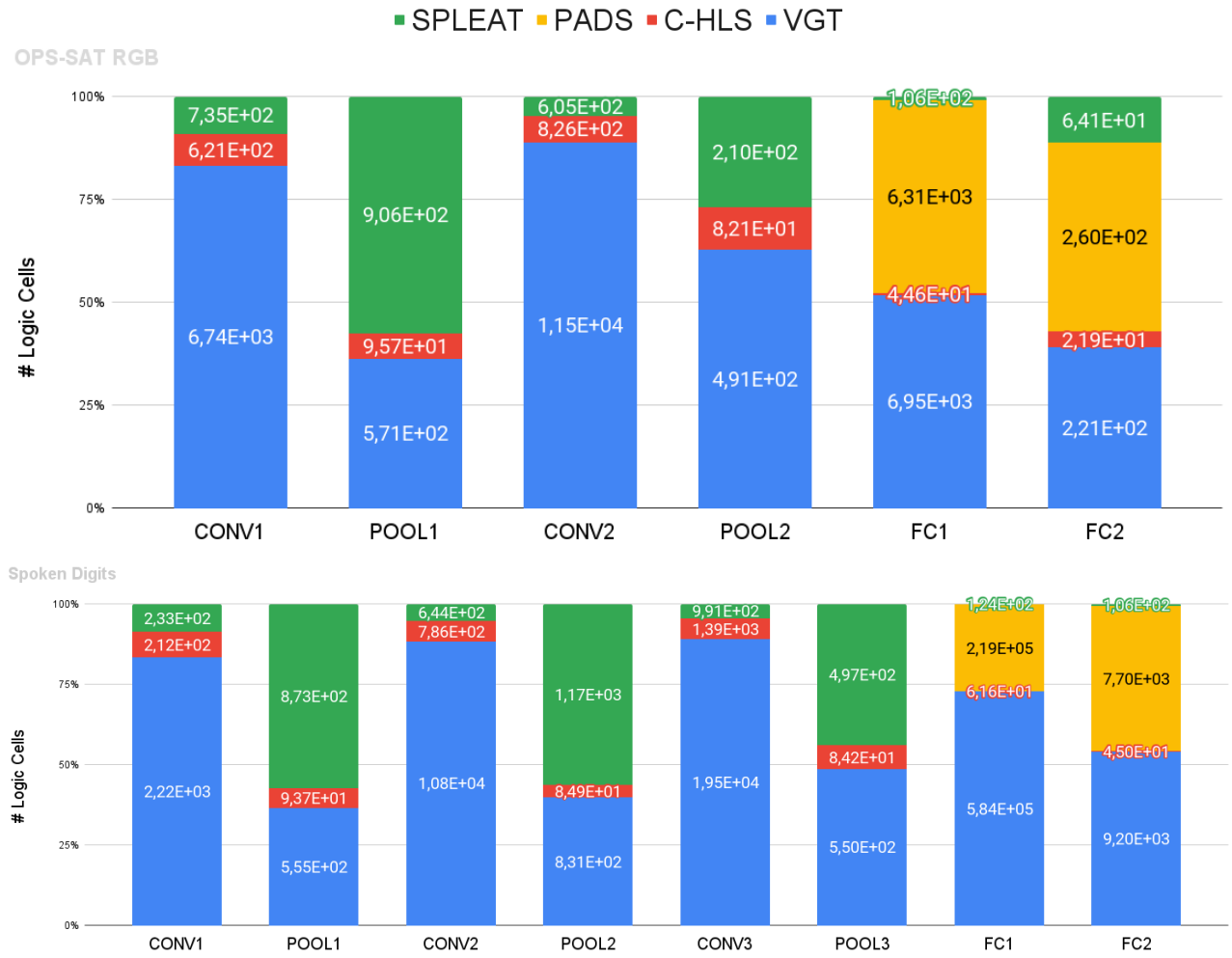


Figure 5.10: Layer-wise LUT estimation in VGT, HLS, PADS and SPLEAT. Top: OPSSAT RGB, bottom: Spoken Digits

5.3.1 Layer-wise estimation

First, we provide layer-wise resource estimations. Figure 5.10 features the layer-wise LUT usage for the four architectures on OPS-SAT and Spoken Digits datasets. It should be noted that FF usage is strongly correlated with LUT usage, thus it is not shown here. The FF usage graph is available in Appendix .2. Figure 5.11 shows the layer-wise RAM and DSP usage.

Estimations yields that $LUT^{Conv}_{SPLEAT} \approx LUT^{Conv}_{C\text{-}HLS}$. However, it also shows that $LUT^{Pool}_{SPLEAT} \approx 1.5 \times LUT^{Pool}_{C\text{-}HLS}$. Therefore, SPLEAT is not suited to max-pooling layers. The estimations are coherent with the literature, as transcoding max-pooling layers towards spiking domain is widely accepted as challenging [106]. In literature, authors propose to incorporate pooling operation in convolution layers. That is, by using a stride of two [14] or specific synaptic weight matrices [107]. Such a mechanism could thus benefit to spiking CNN hardware implementations. Additionally, $LUT^{FC}_{SPLEAT} \approx 2.5 \times LUT^{FC}_{C\text{-}HLS}$. Thus, SPLEAT uses more LUTs in both Pool and FC layers. This is explained by low-level differences between SPLEAT and C-HLS. The latter is automatically generated from synthetisable C code. Pragmas ensure the maximum level of multiplexing during synthesis. SPLEAT on the other hand is designed by hand and does not feature such

Figure 5.11: Layer-wise RAM and DSP estimation in VGT, HLS, PADS and SPLEAT. Top: OPSSAT RGB, bottom: Spoken Digits

low-level optimizations. The difference in implementation choices explains the difference between expectations and estimations. Indeed, the literature reports up to 50% resource savings [88] when using spiking rather than formal acceleration. Figure 5.11 shows that RAM and DSP usage is similar in C-HLS in Convolution and FC layers. Similarly to LUT estimations, that is not true for max-pooling layers. Indeed, C-HLS max-pooling layers mostly use RAM whereas SPLEAT max-pooling layers mostly use DSPs. In both cases, usage remains very low and is far from reaching the board capacity.

The comparison between PADS and VGT (FC layers) yields that $LUT_{PADS}^{OPSSAT\text{-}FC} \approx LUT_{VGT}^{OPSSAT\text{-}FC}$. However, $LUT_{VGT}^{Spoken\text{-}FC1} \approx 2.5 \times LUT_{PADS}^{Spoken\text{-}FC1}$ and 1.2 for FC2. Hence, PADS enables resource savings over VGT for large layer sizes. This is coherent with the 3D plots of LUT interpolation surface shown in Figure 5.3 (first column, second raw), which shows that VGT resource usage increases faster than PADS. The savings offered by PADS are even greater regarding DSP usage. Indeed, VGT uses a lot of DSPs whereas PADS uses none. In Spoken Digits CNN, both FC1 and FC2 reach the maximum DSP capacity of the board on their own (Zedboard has 220 DSPs available). The saturation of DSP by VGT is responsible for the LUT overhead observed in large FC layers. In all, PADS offers 100% reduction of DSP usage which is a critical and scarce resource in FPGAs, and 2.5 reduction of LUT usage in large FC layers. This is thus an interesting

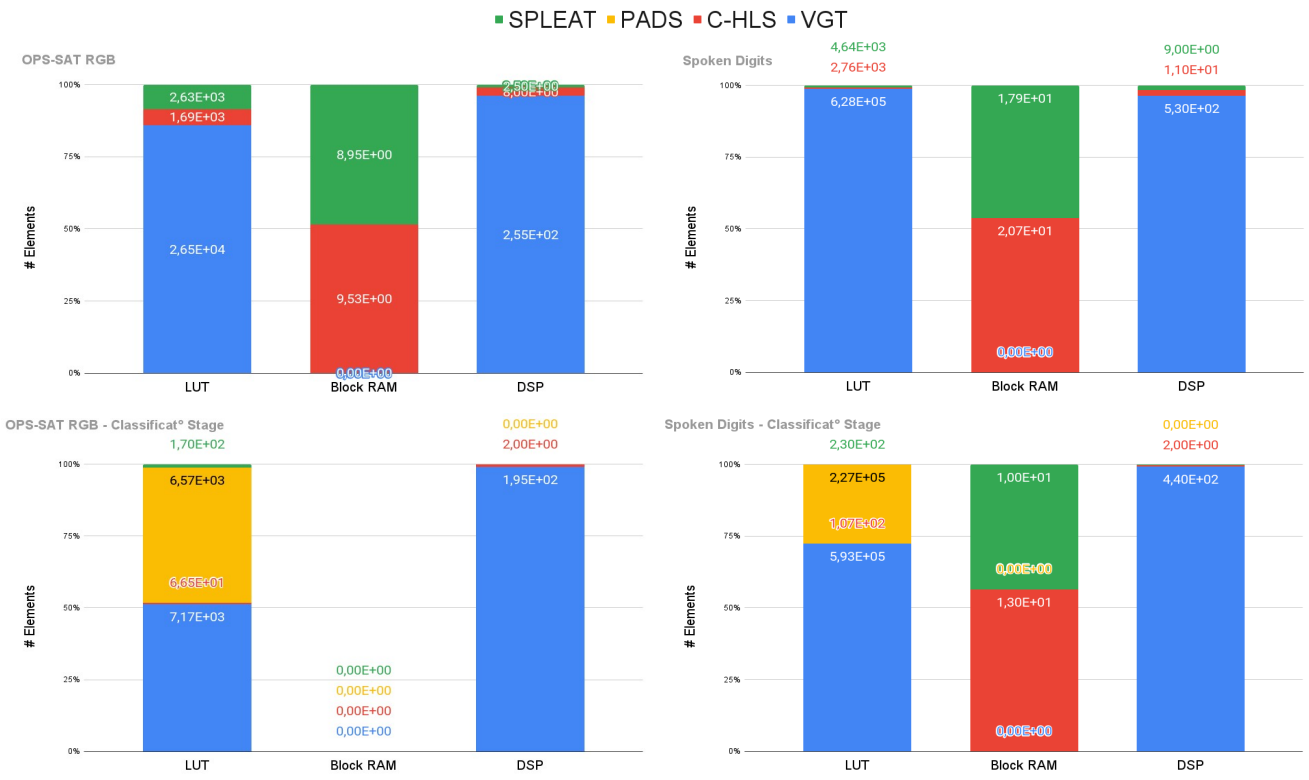


Figure 5.12: Comparisons of resource usage in VGT, HLS, PADS and SPLEAT. Left: OPSSAT, Right: Spoken Digits. Top: Full network, Bottom: Classification Stage

feature for embedded systems in which size is critical.

Lastly, the comparison between PADS and VGT resource estimations yields that SPLEAT LUT usage is far below that of PADS. This is quite straightforward due to the very different level of parallelism. Moreover, PADS resource usage strongly depends on layer size. This is not the case for SPLEAT. Thus, the resource overhead of PADS is greater in larger layers: $LUT_{PADS}^{Spoken\text{-}FC1} \approx 1700 \times LUT_{SPLEAT}^{Spoken\text{-}FC1}$. In the much smaller OPSSAT-FC2 layer, the ratio is only 4. That is, SPLEAT is preferable over PADS regarding LUT usage. As a counterpart, PADS reduces Block RAM usage by 100% compared to SPLEAT. However, SPLEAT BRAM usage is negligible in front of the available resource, so that PADS savings are not significant.

5.3.2 Network-wise estimation

After discussing layer-wise resource results, we propose to compare accelerators at network level. The results provided in Figure 5.12 are obtained by summing the layer-wise estimations. It should be noted that effect of DSP and BRAM saturation is not taken in account here. Therefore, the total DSP and BRAM usage may overpass the board limit in some cases (Zedboard). This approximation is the same for all architectures, thus the results remain comparable. Improvements regarding the acknowledgment of DSP and BRAM saturation will be detailed in the outlooks of this Chapter (Section 5.8).

The top row of Figure 5.12 features the resource usage estimations for full CNNs. PADS does not cover convolution and pooling layer therefore it does not appear in those results. The bottom row features resource estimation for the fully-connected stages including PADS. The network-

wise estimations confirm the conclusions derived from layer-level results. In general, we observe $LUT_{SPLEAT} \approx 2 \times LUT_{C\text{-}HLS}$. The network wise DSP and Block RAM usage are equivalent in C-HLS and SPLEAT in both OPS-SAT and Spoken Digits datasets. The classification-stage level estimations for VGT and PADS also confirms the layer-wise results: PADS offers 50% LUT saving in Spoken Digits, and 100% DSP savings in both datasets. PADS and VGT feature very similar low-level implementation choises, except for the coding domain. Hence, those results confirm that spiking domain provides reliable resource savings in highly parallel architectures, especially for large network sizes. Lastly, the results also confirm the conclusions derived from the comparison of SPLEAT and PADS at layer level. That is, SPLEAT uses much fewer resources than PADS, especially in larger FC layers.

5.3.3 Conclusions

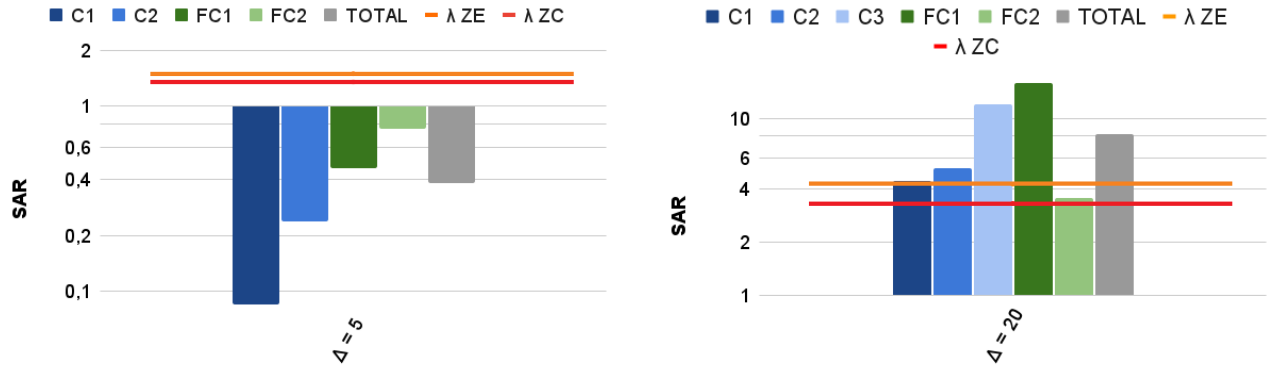
As explained earlier, fully-parallel implementations are unrealistic for state-of-the-art CNN implementations. For simple tasks implying small topologies, high-level of parallelism is however conceivable. In this context, spiking implementations could bring substantial resource savings. That is mitigating size and weight in the system. In satellite applications such as OPS-SAT cloud segmentation [93], small CNNs are sufficient for acceptable accuracy, and size is one of the most limiting factors. Such application could therefore benefit from spiking highly-parallel implementations instead of formal ones. The same conclusions can be made for other niches applications, like object tracking in drones [108] also involves small CNNs and drastic size constraints. Such applications could also benefit from spiking highly-parallel acceleration. In both cases, one might expect to reduce LUT usage by 50% and DSP usage by 100%. Thus, smaller FPGAs can be used for the same task, reducing weight and size of the system. Furthermore, the resource estimations have confirmed that max-pooling operation is challenging for spiking domain. Indeed, SPLEAT uses ten times more resources than C-HLS for max-pooling layers. Such layers might be integrated to convolution operations as proposed in literature [14] [106].

5.4 Inference time and power estimations

In this section, we describe the inference time and power estimations for the four accelerators on OPS-SAT and Spoken Digits datasets. Like for other results, the estimations for all datasets of the benchmark are available in Appendix .2. Like for resources, we begin with layer-wise estimations and give network level results afterwards. The same notations are used: $Metric^{Obj}_{Arch}$, where $Metric$ can be either T for time and P for power consumption. Additionally, DP and SP stands for dynamic and static power respectively. Like for resource results, estimations are discussed by comparing SPLEAT to C-HLS, PADS to VGT and SPLEAT to PADS. Before going into further details, the layer-wise and total SAR obtained on OPS-SAT and Spoken Digits are reminded in Figure 5.13. For OPS-SAT, formal test accuracy is 91.2% and spiking test accuracy is 91.1% ($\Delta = 4$). For Spoken Digits, formal accuracy is 88.55% and spiking is 87.5% ($\Delta = 20$).

5.4.1 Layer-wise estimation

The layer-wise inference time estimations are depicted in Figures 5.14. For spiking accelerators, the results are averaged on 100 images. Indeed execution time of spiking accelerators depends on



(a) (b)

Figure 5.13: Layer-wise and total SAR for a) OPS-SAT ($\Delta = 4$) and b) Spoken Digits ($\Delta = 20$).

synaptic activity, *i.e.* on the input sample. The estimations are obtained with MinPeriod = 1 timesteps and MaxPeriod = 1000 timesteps.

As expected, C-HLS is much slower than SPLEAT for convolutions and FC layers. C-HLS convolutions are a hundred to a thousand times slower than SPLEAT. For FC layers, that is three to a hundred times slower. That is the counterpart of the resource savings brought by the intense multiplexing of C-HLS architecture. It should be noted that SPLEAT inference time is affected by SAR. Indeed the ratio of inference time $\frac{T^{Conv\&FC}_{C\text{-}HLS}}{T^{Conv\&FC}_{SPLEAT}}$ is greater for layers with low SAR (Figure 5.13). In other words, SPLEAT convolutions and FC are always faster than C-HLS and even more for low layer-wise SAR. Thus, the estimations confirms that SPLEAT inference time is coherent with layer-wise SAR. For max-pooling layers, SPLEAT is not always faster: On Spoken Digits, SPLEAT is three to ten times slower than C-HLS. This observation confirms the incompatibility of max-pooling operation and spike coding.

The comparison between PADS and VGT yields that PADS is slower than VGT in all studied layers. In both FC1 layers, PADS is two times slower. In FC2 layers, that is five times. Therefore, the resource savings of PADS FC layers are made at the expense of inference time. In Figure 5.13, $SAR^{OPSSAT\text{-}FC} < 1$. That is, there are fewer synaptic operations in the SNN than there are in the FNN. In the light of this metric, one might expect PADS to be faster in OPS-SAT FC layers. However, the temporal sparsity of spikes is not taken in account in SAR. This sparsity is intrinsic to rate encoding but strengthen by the Spike Generation Cell design. In consequence, PADS is slower than VGT even when SAR<1. Moreover, this time the inference time ratios are not consistent with SAR trends. That is, $\frac{T^{Spoken\text{-}FC2}_{PADS}}{T^{Spoken\text{-}FC2}_{VGT}}$ is greater than $\frac{T^{Spoken\text{-}FC1}_{PADS}}{T^{Spoken\text{-}FC1}_{VGT}}$ although $SAR^{Spoken\text{-}FC1} > SAR^{Spoken\text{-}FC2}$. That is also caused by the temporal sparsity of spikes, which is non deterministic and not taken in account by the SAR metric.

The comparison between PADS and SPLEAT yields interesting results to understand the influence of parallelism on rate-coded SNN accelerators. Indeed, SPLEAT and PADS have similar inference time for all FC layers except Spoken-FC1. That is surprising, as the parallelism and pipeline of PADS should enable a drastic speed-up. As shown in Figure 5.13, $SAR^{Spoken\text{-}FC1}$ is a lot higher than in other layers. When SAR is low, SPLEAT is able to process spikes on-the-fly, *i.e.* before the next spike arrives. In such condition, the parallelism and pipeline of PADS are useless, and the two architectures have the same inference time. On the other hand, things changes for higher SAR since spikes arrive much more frequently or even simultaneously at the input of

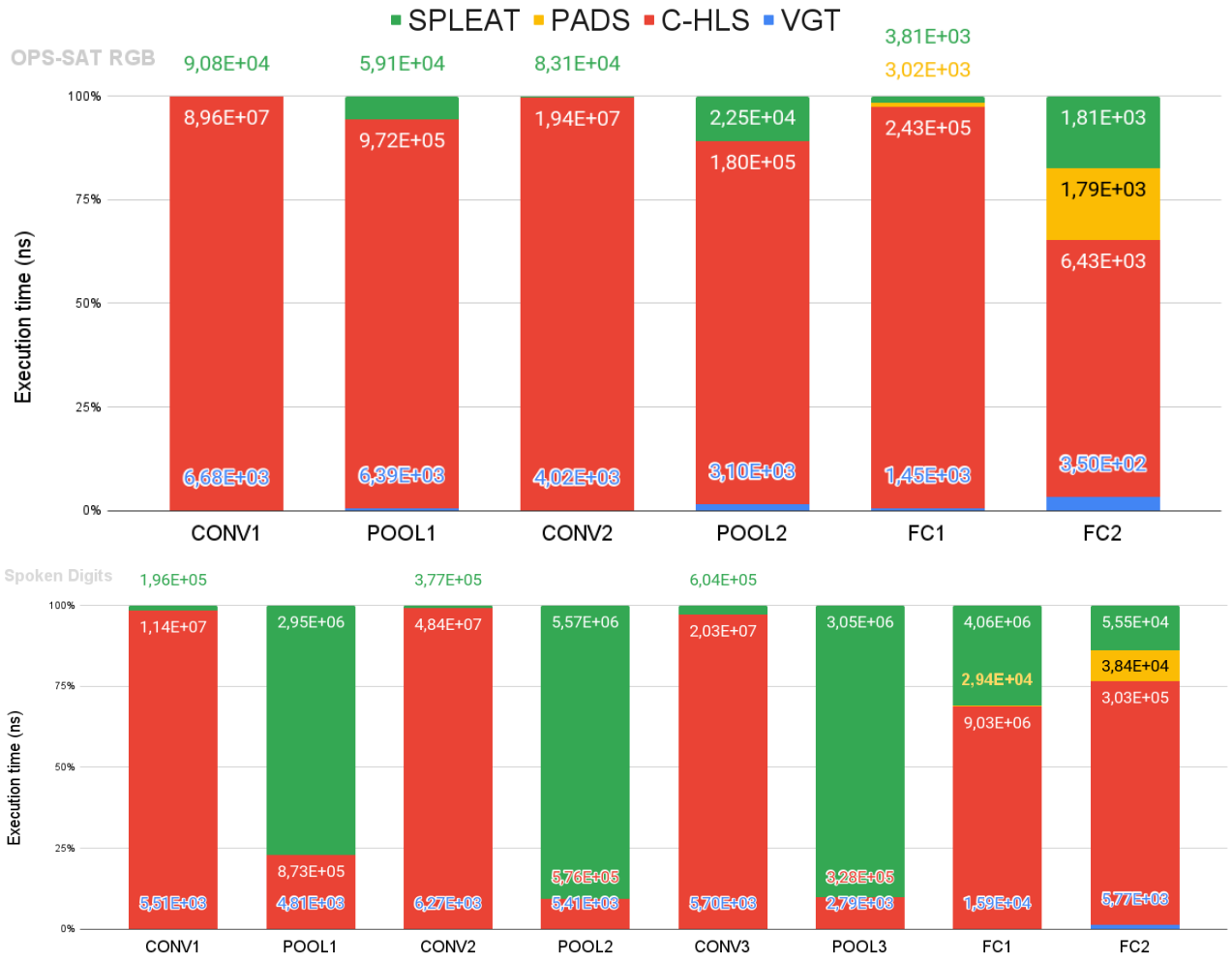


Figure 5.14: Layer-wise execution time (per image) estimation in VGT, HLS, PADS and SPLEAT. Top: OPSSAT RGB, bottom: Spoken Digits

the layer. Thus, SPLEAT is not able to process spikes as they arrive, hence the observed time overhead. Under such conditions, the parallelism of PADS is better exploited and the architecture is much faster than SPLEAT. This observation deeply strengthen the role of the SAR metric: it can be used to tune parallelism at layer level in spiking architectures. Using lower level of parallelism when SAR is low can bring substantial resource savings (Section 5.3) without time overhead.

The layer-wise dynamic power estimations are provided in Figure 5.15. At this stage, only dynamic power is taken in account. Static power only makes sense at network level: it is related to the whole design and board. Estimations demonstrates that SPLEAT dynamic power usage is lower than C-HLS in all layers of both CNNs, except Spoken Digits-FC1. That is, even in max-pooling layers which caused resource and time overhead. DP_{SPLEAT} is 2 to 4 times lower than $DP_{C\text{-}HLS}$, except in Spoken Digits FC1 where $DP_{SPLEAT} \approx 2 \times DP_{C\text{-}HLS}$. Overall, SPLEAT provides greater power savings over C-HLS when SAR is low. This is coherent with previous observations on inference time: a low SAR means that SPLEAT latency is masked by the temporal sparsity of spikes. In such conditions, SPLEAT is often idle and waiting for inputs. On the other

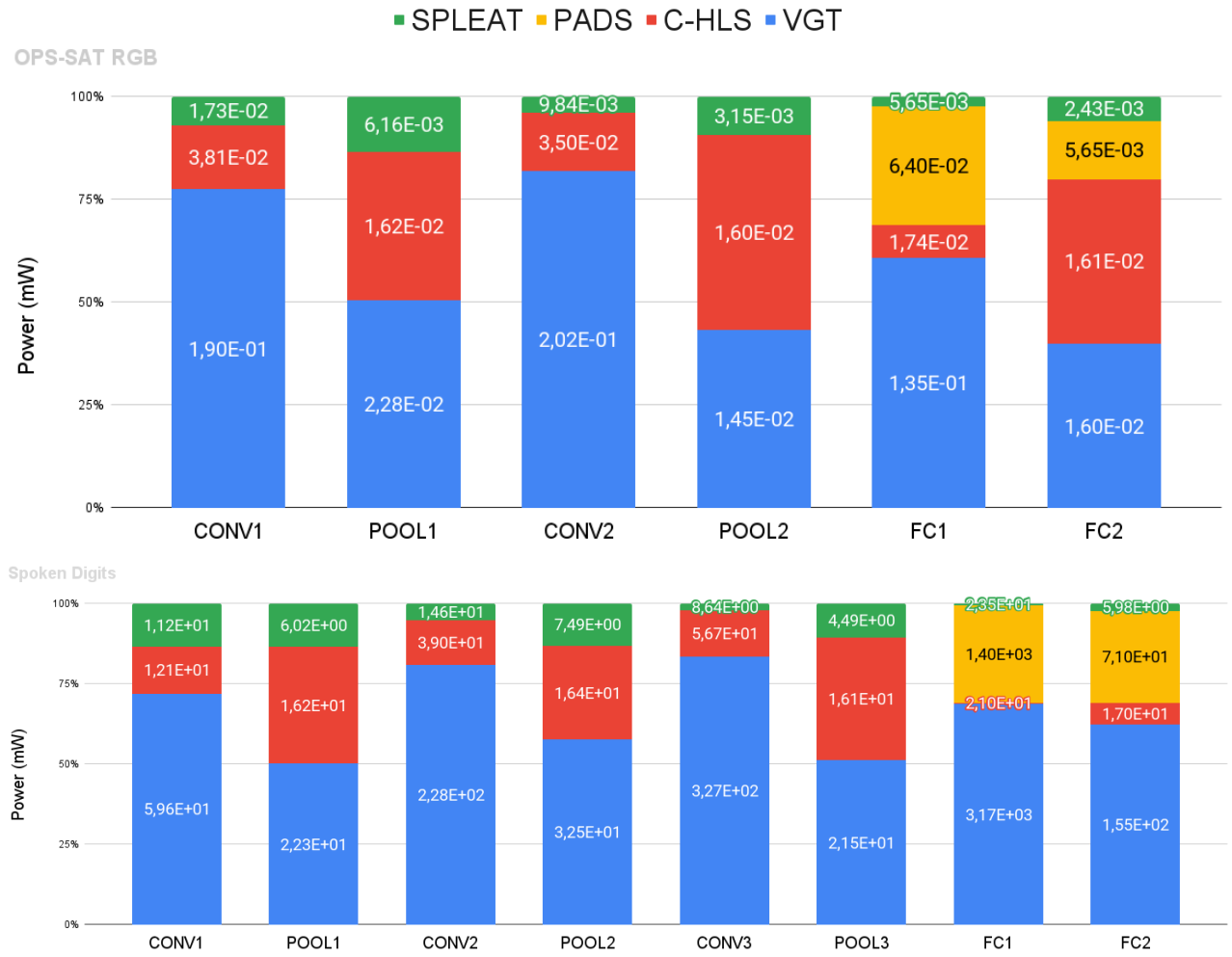


Figure 5.15: Layer-wise Power estimation in VGT, HLS, PADS and SPLEAT. Top: OPSSAT RGB, bottom: Spoken Digits

hand, C-HLS processes pixels without interruption, thus the dynamic power is higher. Additionally, C-HLS power overhead in pooling-layers can be attributed to the Block RAMs. In all, highly sequential spiking accelerators provides substantial power savings over formal ones. Those savings are correlated to layer-wise SAR.

The second comparison yields that PADS provides significant dynamic power savings over VGT. That is, DP_{PADS} is two three times smaller than DP_{VGT}. That is, the time overhead of highly-parallel spiking accelerators is compensated by dynamic power savings. Moreover, the power savings are also correlated with SAR. That is, $\frac{DP_{VGT}}{DP_{PADS}}$ follows the same trend as SAR.

Lastly, the comparison between SPLEAT and PADS confirms the discussions on computation density. SPLEAT dynamic power usage is drastically lower than PADS as expected from the parallelism. Moreover, $\frac{DP_{PADS}}{DP_{SPLEAT}}$ is higher for higher SAR: 60 in Spoken-FC1 (SAR=16) and only 2.3 in OPSSAT-FC2 (SAR=0.8). The computation density explains such results: as explained, PADS processes spikes in parallel when SAR is high. Such high computation density increases dynamic power usage, which is visible in the results. This observation strengthen our statement that the SAR metric is a crucial tool for layer-level tuning of parallelism in spiking accelerators.

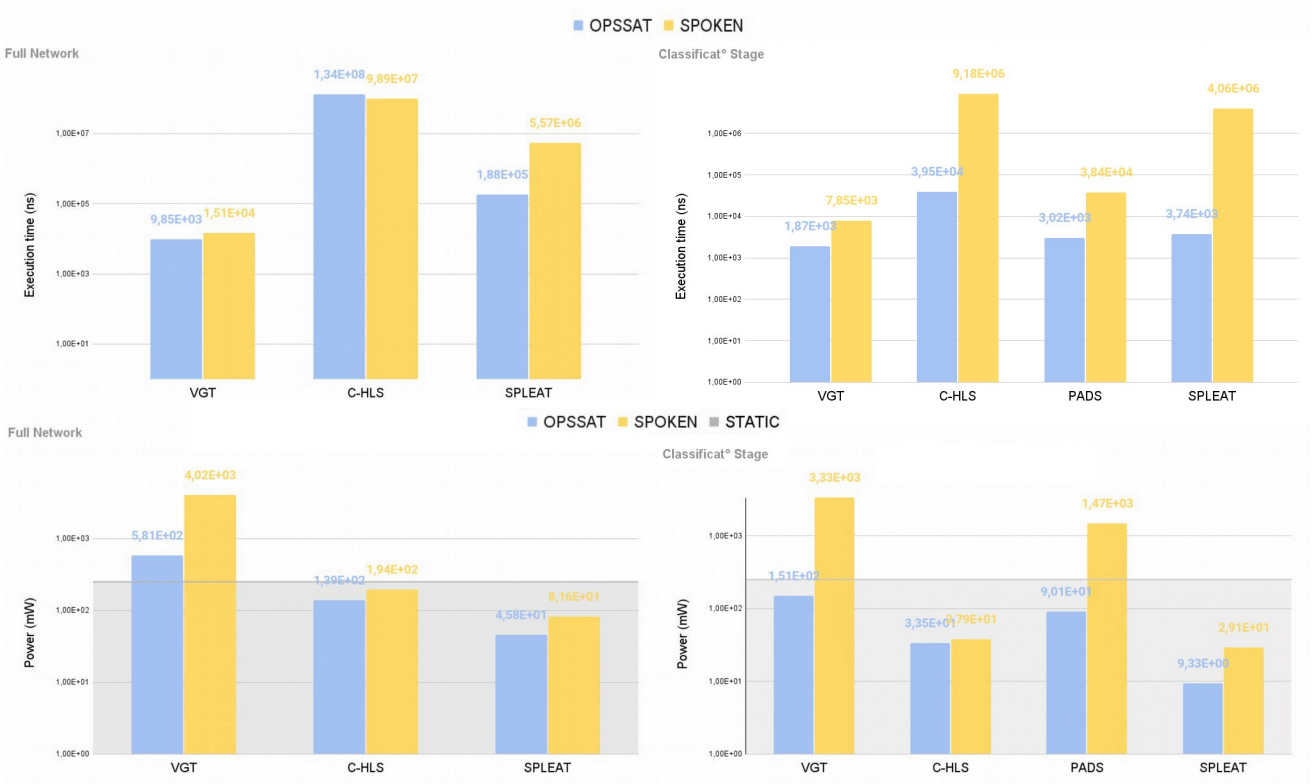


Figure 5.16: Comparisons of inference time (per image) and power usage in VGT, HLS, PADS and SPLEAT. Left: full network, Right: classification stage. Top: time, Bottom: power

In doing so, one is not only able to optimize inference time for resource usage, but also dynamic power.

5.4.2 Network-wise estimation

After studying layer-wise results, inference time and power estimations are provided at network level in Figure 5.16.

Network-wise inference time estimations are depicted in the top row of Figure 5.16. Like for resource results, the figure is divided between full CNN estimations (left) and classification stage estimations (right). Inference time estimations are discussed first and power will be addressed after. The comparison of SPLEAT and C-HLS yields that SPLEAT is even faster at network-level. Indeed, SPLEAT features an inter-layer pipeline whereas C-HLS does not. SPLEAT is 13,000 times faster on OPS-SAT, and 6,500 times on Spoken Digits. The trend is consistent with SAR and previous observations: SAR is higher in Spoken Digits, thus spike density is higher and SPLEAT is not able to process all spikes on-the-fly. The comparison between PADS and VGT are also consistent with layer-wise results. On OPS-SAT PADS classification stage is 1.5 times slower than VGT. On Spoken Digits that is 4.9 times. Like for sequential architectures, this is coherent with the SAR metric. Lastly, the comparison of PADS and SPLEAT yields that SAR is also interesting for tuning parallelism at network-level. Indeed, for OPS-SAT network-level times are equivalent. For Spoken Digits where SAR is higher, PADS is a hundred time faster. In such context parallelism is required to mitigate latency in high SAR applications but useless otherwise.

The network-wise power estimations are depicted in the bottom row of Figure 5.16. In contrast

with layer-wise results, the graphs feature the static power of the Xilinx Zedboard (0.250 W). Like for time estimations, the network-level power is coherent with aforementioned layer-wise results: spiking domain enables substantial power savings over formal domain. SPLEAT reduces dynamic power by a factor two compared to C-HLS, and PADS enables 50% savings over VGT. Is should be noted that PADS power savings are slightly improved in OPS-SAT, *i.e.* low SAR. However, the savings are negligible in front of the common static power offset. That is, the overall power usage (dynamic+static) will remain unchanged. This is true for FPGA where static power is high. However, static power is much lower in ASICs. Such systems could therefore highlight the power savings provided by spiking domain. Lastly, the comparison between PADS and SPLEAT is consistent with the layer-level discussions: SAR is higher in Spoken Digits, thus computation density in PADS is higher. That explains the increased power consumption difference in Spoken Digits. However, the differences are also negligible in front of static power.

5.4.3 Conclusions

The study of time and power estimations yields several strong conclusions. The layer-wise results confirmed that max-pooling layers are challenging for spiking domain. SPLEAT max-pooling layers indeed suffer severe latency overhead compared to C-HLS when SAR is high. That is, despite the higher level of multiplexing found in C-HLS architecture. However, SPLEAT is much faster than C-HLS overall (factor 6500 to 13000), all while reducing dynamic power usage by a factor two. The differences in processing time is mostly due to low-level implementation choices, and the power savings are small in front of the static power offset, which mitigates the proposed results. On the other hand, PADS is slower than VGT (factor 1.5 to 5) due to the temporal sparsity of spikes. This temporal sparsity is intrinsic to rate encoding, but increased by the current sequential spike generation process. Reducing spike temporal sparsity is therefore crucial to compete with highly parallel formal accelerators. Such improvement will be detailed in the outlooks of this chapter. Regarding power, PADS also provides savings which are small in front of the static power offset.

Moreover, the results discussed in this section strengthen the role of the SAR metric. Indeed, it can be used to tune parallelism at both layer and network-level. That is, high level of parallelism should be used to mitigate timing when SAR is high. Otherwise, parallelism is useless and does not provide speed-up. For example, using SPLEAT instead of PADS in OPSSAT Classification stage does not imply latency overhead, but provide 90% savings in resource and dynamic power. Realistic architectures should feature intermediate level of parallelism, which can be tuned using SAR. Further work should be undertaken to quantify the relation between SAR and parallelism in spiking accelerators.

5.5 Energy estimations

Finally, timing and power are combined to provide energy estimations ($E = P \times \Delta T$). In this section, we proceed like for other estimations: layer-wise estimations are addressed first, followed by network-level results. Energy is widely accepted as the most important and universal limiting factor for embedded systems. The results of this section are thus crucial for the cartography of applications and neural coding domains. In this section, the same notations are used to simplify explanations. The estimations are noted E^{Obj}_{Arch}, where E stands for energy consumption. It should be noted that static power is taken in account in network-wise energy estimations, but not in layer-wise.

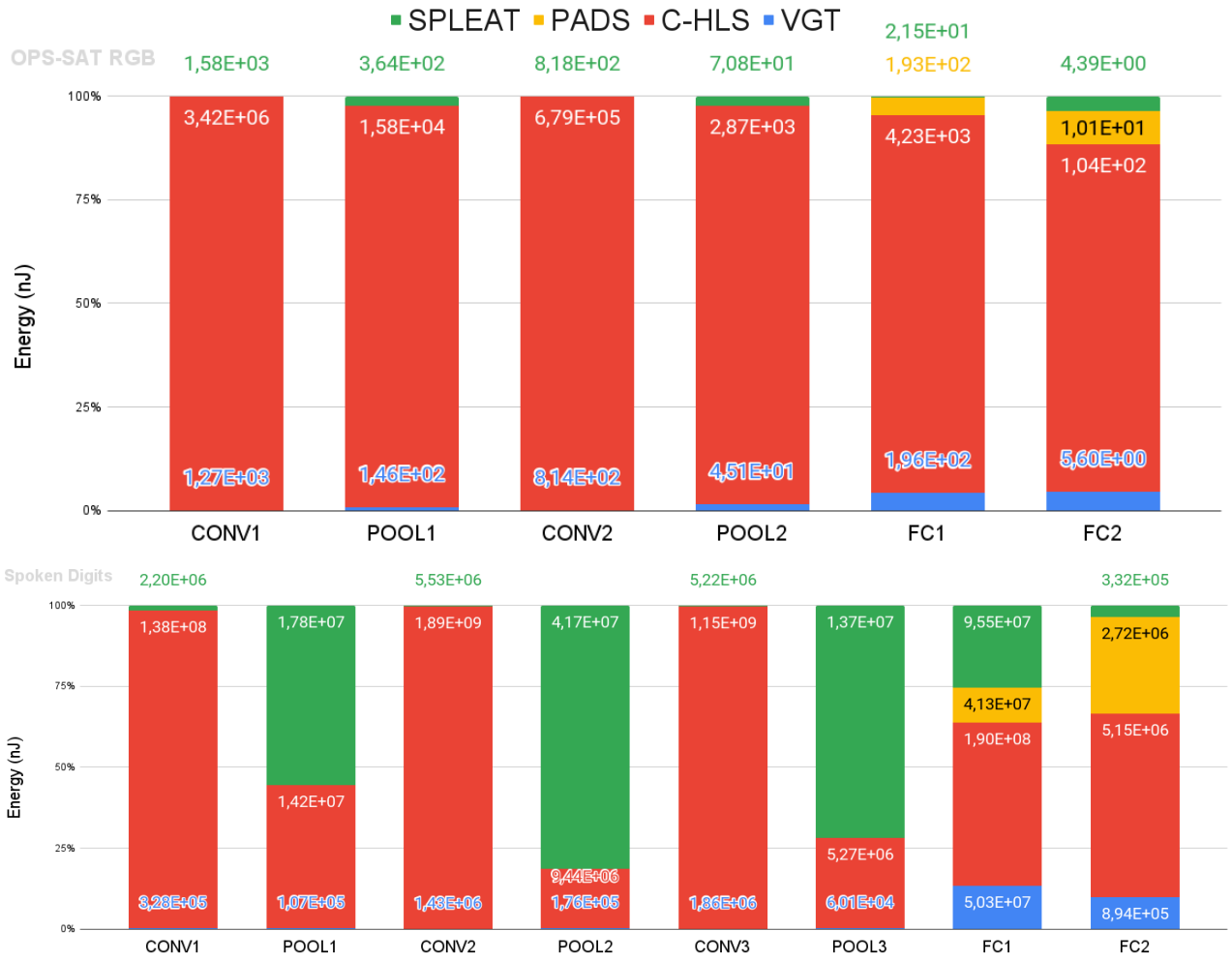


Figure 5.17: Layer-wise energy estimation (per image) in VGT, HLS, PADS and SPLEAT. Top: OPSSAT RGB, bottom: Spoken Digits

5.5.1 Layer-wise estimation

Layer-wise energy estimations for VGT, C-HLS, PADS and SPLEAT on OPS-SAT (above) and Spoken Digits (below) datasets are provided in Figure 5.17. The comparison of SPLEAT and C-HLS yields that the first provide drastic power savings in Conv and FC layers. That is, E^{Conv}_{SPLEAT} is forty to two thousand times lower than $E^{Conv}_{C\text{-}HLS}$. In FC layers, energy is reduced by a factor 2 to 200. Moreover, SPLEAT energy savings are higher for low layer-wise SAR. That is, $\frac{E^{Conv\ \&\ FC}_{SPLEAT}}{E^{Conv\ \&\ FC}_{C\text{-}HLS}}$ is loosely proportional to the layer-wise SAR. On the other hand, pooling layers imply an energy consumption overhead in Spoken Digits CNN. Thus, the power savings observed in SPLEAT pooling layers over VGT are not significant enough to compensate the latency overhead. Those estimations confirm that max-pooling layers should be avoided in spiking architectures, particularly when SAR is high.

In highly parallel architectures, the layer-wise energy savings provided by PADS is less straightforward. According to Figure 5.17, PADS and VGT have nearly identical energy consumption in $FC1^{\text{OPS-SAT}}$. In $FC2^{\text{OPS-SAT}}$, PADS uses 2 times more energy than VGT. Hence PADS does

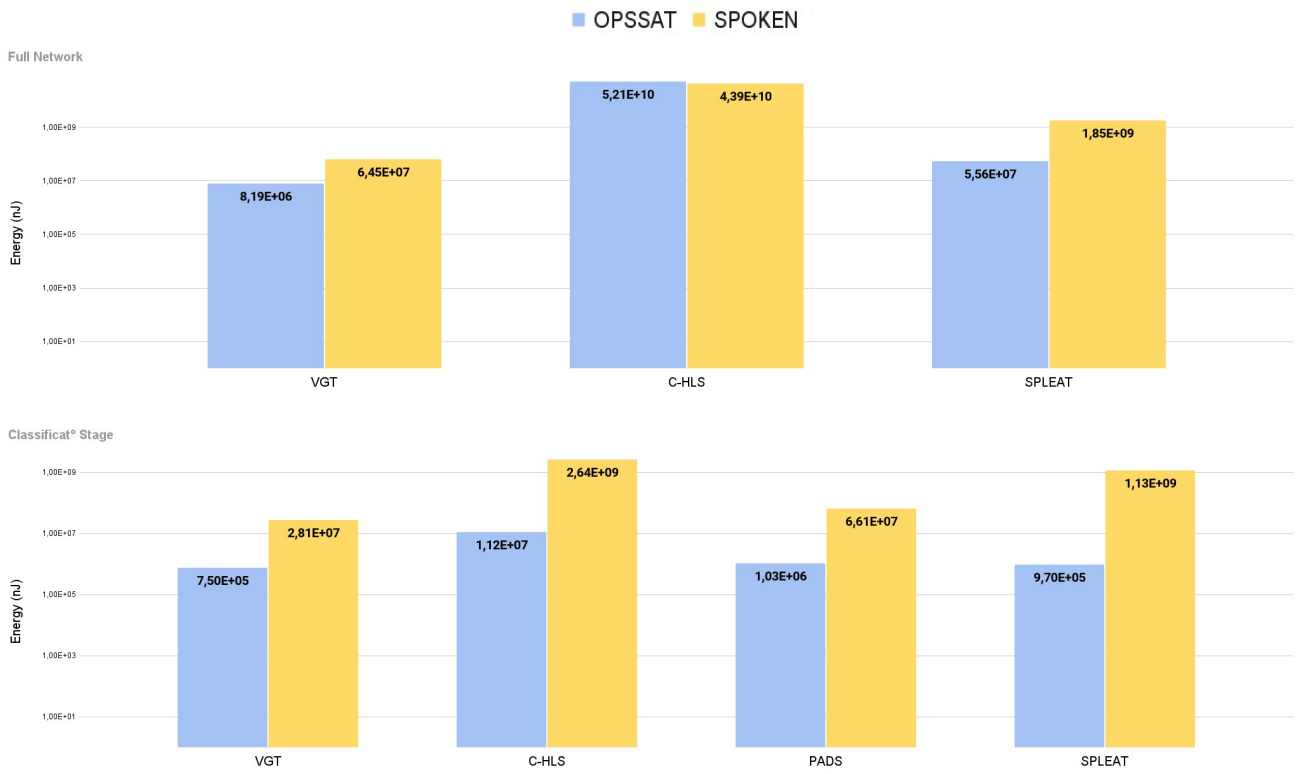


Figure 5.18: Comparisons of energy consumption per image in VGT, HLS, PADS and SPLEAT. Left: full network, Right: classification stage. Top: power, Bottom: timing

not provide reliable-energy savings over VGT despite the low layer-wise SAR measured in OPS-SAT. In Spoken Digits, $E^{\text{Spoken-FC1}}_{\text{PADS}}$ is 20% lower than $E^{\text{Spoken-FC1}}_{\text{VGT}}$, whereas $E^{\text{Spoken-FC2}}_{\text{PADS}}$ is three times higher than $E^{\text{Spoken-FC2}}_{\text{VGT}}$. More than a confirmation that PADS does not provide reliable energy savings at layer-level, this second result is incoherent with SAR measurements. That is, $SAR^{Spoken-FC1} > SAR^{Spoken-FC2}$, but $\frac{E^{Spoken-FC1}_{PADS}}{E^{Spoken-FC1}_{VGT}} < \frac{E^{Spoken-FC2}_{PADS}}{E^{Spoken-FC2}_{VGT}}$. Therefore, the SAR is not correlated to layer-wise energy ratios in highly-parallel architectures.

Lastly, the layer-wise comparison of PADS and SPLEAT yields that the first is only preferable to the second when SAR is high, in terms of energy consumption. When SAR is low (OPSSAT-FC1&FC2 and Spoken-FC1), SPLEAT consumes 2.3 to 9 times fewer energy than PADS. On Spoken-FC2 on the other hand (high SAR), SPLEAT consumes 2.3 times more energy than PADS. This observation confirms the role of SAR in determining parallelism at layer-level for resource, time, power and energy optimization.

5.5.2 Network-wise estimation

The network-level estimations are provided in Figure 5.18. Like for previous estimation metrics, the results are split between full-network (top) and classification stage (bottom). The comparison between C-HLS and SPLEAT confirms that the latter provides drastic energy savings over the first. That is, $E^{\text{OPSSAT}}_{\text{HLS}} \approx 1000 \times E^{\text{OPSSAT}}_{\text{SPLEAT}}$ and $E^{\text{Spoken}}_{\text{HLS}} \approx 25 \times E^{\text{Spoken}}_{\text{SPLEAT}}$. Those ratios are consistent with the network-level SAR measurements (0.4 in OPS-SAT and 8.3 in Spoken-Digits). That is, SPLEAT advantage is greater for low SAR.

In the field of highly parallel architectures, PADS uses more energy than VGT in both datasets. More precisely $E_{\text{PADS}}^{\text{OPSSAT}} \approx 1.3 \times E_{\text{VGT}}^{\text{OPSSAT}}$ and $E_{\text{PADS}}^{\text{Spoken}} \approx 3 \times E_{\text{VGT}}^{\text{Spoken}}$. Those ratios are consistent with the network-wise SAR, and indicates that PADS energy overhead is mitigated for low SAR. That is, the estimations confirm our hypothesis that spiking domain is better adapted to low SAR. The energy overhead is due to the inference time overhead in PADS, which is caused by the temporal sparsity of spike encoding. Hence, reducing temporal sparsity of spike encoding is crucial to compete with formal implementation in terms of energy. Moreover, those results are not consistent with the expected cartography based on SAR study (Section 4.4). The correlation between estimation-based cartography and SAR-based cartography will be discussed in details in Section 5.6, for both high and low level of parallelism.

Finally, the comparison between PADS and SPLEAT confirms the role of SAR in tailoring parallelism for spiking implementations. Indeed, SPLEAT uses fewer energy than PADS on OPSSAT where SAR is low (10%), but more on Spoken Digits where SAR is high (17 times more). That is, it is preferable to use SPLEAT than PADS on OPSSAT. That is not only for the slight energy savings but also for the drastic resource savings provided by SPLEAT over PADS. On the other hand, when SAR is high like in Spoken Digits, higher level is necessary to mitigate latency and energy.

5.5.3 Conclusions on energy estimations

Energy is widely accepted as the most limiting factor for embedded systems. The conclusions derived from the energy estimations are thus crucial for the cartography of applications and coding domains. The network-level estimations have shown that SPLEAT provides drastic energy savings over C-HLS, regardless of SAR. However, SPLEAT is even more advantageous for low SAR. In such conditions, energy consumption is reduced by a factor 1000. However, the conclusions are mitigated by the difference of low-level implementation in the two sequential architectures. The comparison is not exactly made with all other things equal, thus the savings might also come from architectural choices. Further experiments should be undertaken with more comparable sequential accelerators to confirm our findings. On the other hand, VGT and PADS are much more similar to each other, increasing our confidence in the following results. That is, PADS uses 1.3 to 3 times more energy than VGT on OPSSAT and Spoken Digits respectively. This overhead is caused by the temporal sparsity of spikes, due to the rate-encoding itself but also to the Spike Generation Cell implementation. This issue should be addressed urgently to leverage spiking domain energy benefits. For both datasets, the SAR-based cartography proposed in Section 4.4 is partially contradicted by energy estimations. This issue will be addressed in the following section (5.6).

Concerning layer-wise estimations, the results confirmed that max-pooling layers are challenging for SNNs, and must be avoided in spiking hardware implementations. That is, by using higher strides in convolution as proposed in the literature [14] [106]. Moreover, the results confirmed the role of SAR in tailoring parallelism, whether at layer or network level. Indeed, the highly parallel design of PADS is underused in low SAR cases, which creates unnecessary resource and power overhead without reducing inference time. On the other hand, parallelism is necessary to mitigate latency when SAR is high.

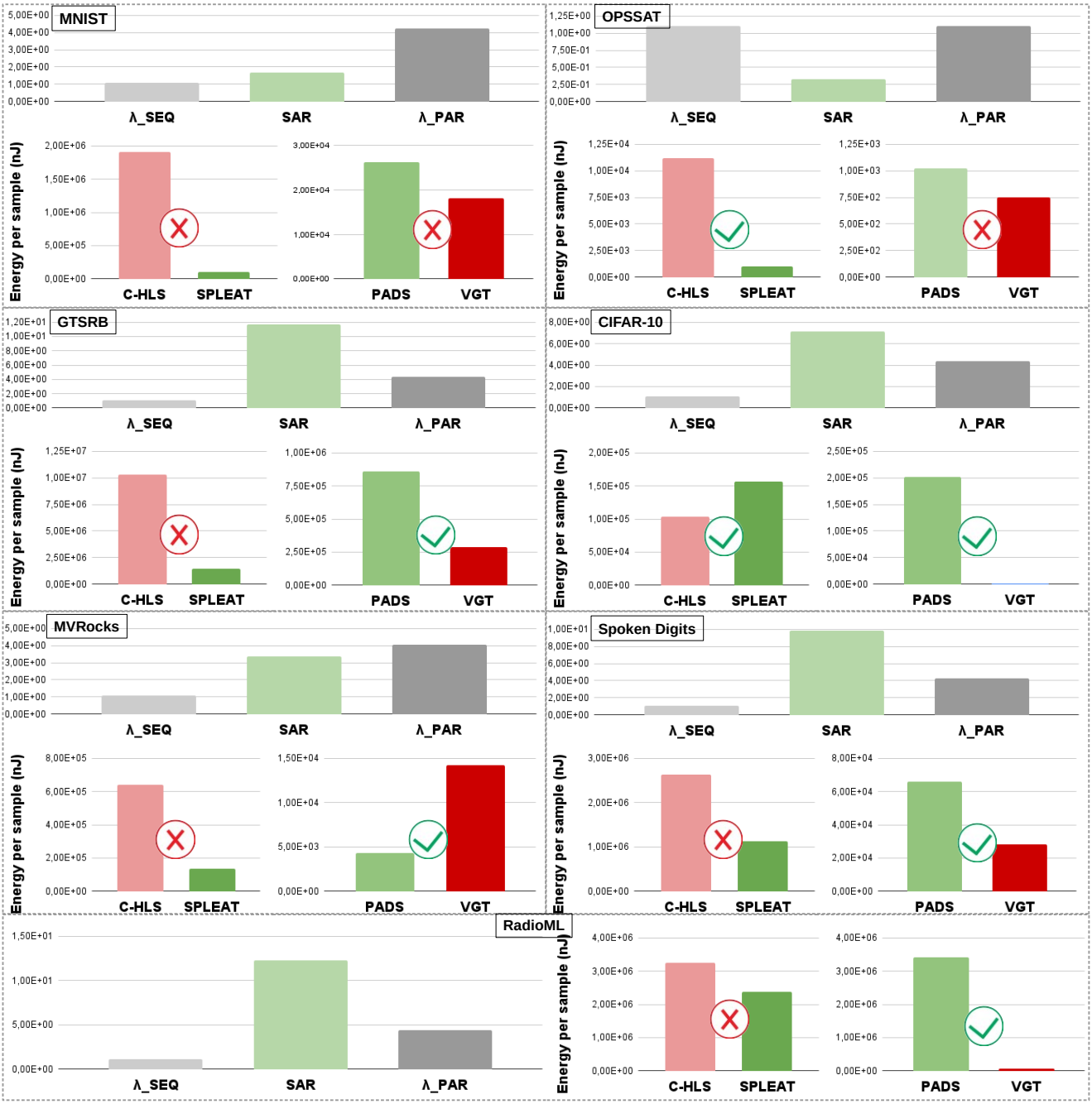


Figure 5.19: Confrontation of SAR model and energy estimations for 7 datasets of the benchmark: MNIST, OPSSAT, GTSRB, CIFAR-10, Mines VS Rocks, Spoken Digits and RadioML 2018. Validation is made separately for parallel and sequential architectures. Each subfigure is divided between SAR & λ values (top) and energy estimations (bottom).

5.6 Validation of the SAR model

One of the goals of this chapter is also to confront the SAR energy model with energy estimations. In doing so, the goal is to evaluate the reliability of the metric, and its usability. More specifically, SAR does not take low-level implementation choices into account such as parallelism or pipeline. As a reminder, the SAR energy model intends to determine whether an application is suitable to spiking acceleration. It is interpreted as a ratio of energy consumption (modulo a constant λ) between equivalent formal and spiking applications. The model is reminded in Equation 5.4.

$$\begin{aligned} \text{SAR} &= \frac{E_{\text{SNN}}}{E_{\text{FNN}}} \times \lambda, \\ \lambda &= \frac{E_{\text{MAC}}}{E_{\text{ACC}}} \end{aligned} \tag{5.4}$$

Where E_{SNN} and E_{FNN} are the energy consumption of equivalent spiking and formal accelerators, and E_{MAC} and E_{ACC} are the energy consumption of MAC and ACC accelerators. The value of λ depends on the device and on the number of MACs in the formal design, as explained in Section 4.3.2 (Equation 4.8). In this case, λ is computed for the Xilinx Zedboard (Zynq-7020). It depends on the design, thus it is computed for each task of the benchmark separately with parallel (PADS/VGT) and sequential (SPLEAT/C-HLS) architectures. According to the model, spiking domain enables energy savings when SAR$> \lambda$ and *vice versa*. The SAR and λ values are shown alongside energy estimations in Figure 5.19. To allow comparison with PADS, all results are limited to the classification stage. The full-network comparisons for SPLEAT and C-HLS are shown in Appendix .2.

In the Figure, cases where the SAR model is correlated with energy estimations are marked with a green tick. Otherwise, graphs are marked with a red cross. For parallel implementations, the SAR model is accurate in five cases out of seven. For sequential implementations on the other hand, the model is wrong in five cases out of seven. A simple explanation can be given for this difference. PADS and VGT have very similar implementations (*i.e.* same level of parallelism and same intra and inter-layer pipeline). On the other hand, we have already explained the low-level differences between SPLEAT and C-HLS. The SAR model is made to compare equivalent architectures thus it is not fully adapted for C-HLS and SPLEAT. Thus, the SAR-based cartography is sensitive to low-level implementation differences.

On the other hand the model is more accurate for the comparison of VGT and PADS. The model still fails in two cases: MNIST and OPSSAT. There are three points which explain the incorrectness of the model on those two cases. First, energy ratio between PADS and VGT is smaller in MNIST and OPSSAT than in other tasks. Hence, the difference of energy consumption might fall within the margin of error of our model. Second, the SAR model does not acknowledge the temporal sparsity of spikes. The sparsity has a strong influence on inference time and therefore energy consumption, which is not represented in the SAR model. Those two first explanations will serve for future improvements of the SAR model. Third, the SAR model is based on the design activity (*i.e.* dynamic power) and neglects the static component. In FPGAs, static power is high and often represents in a significant proportion of the overall power usage. In ASICs on the other hand, the static component is much less significant. In all, the SAR model seems better suited to ASIC implementation than FPGAs. Figure 5.20 shows the energy estimations for MNIST and OPS-SAT when neglecting the static power consumption. In this context, the model is more accurate: the prediction becomes valid for OPSSAT, but still fails on MNIST.

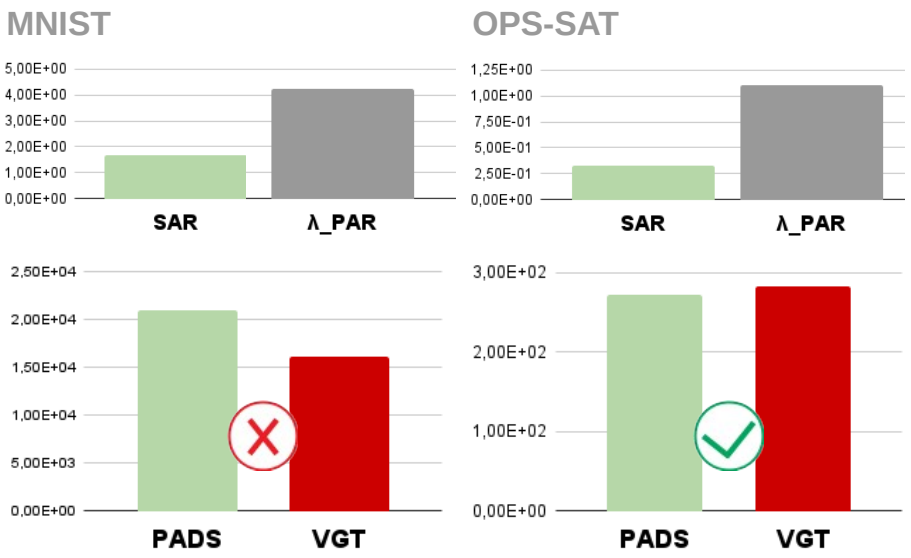


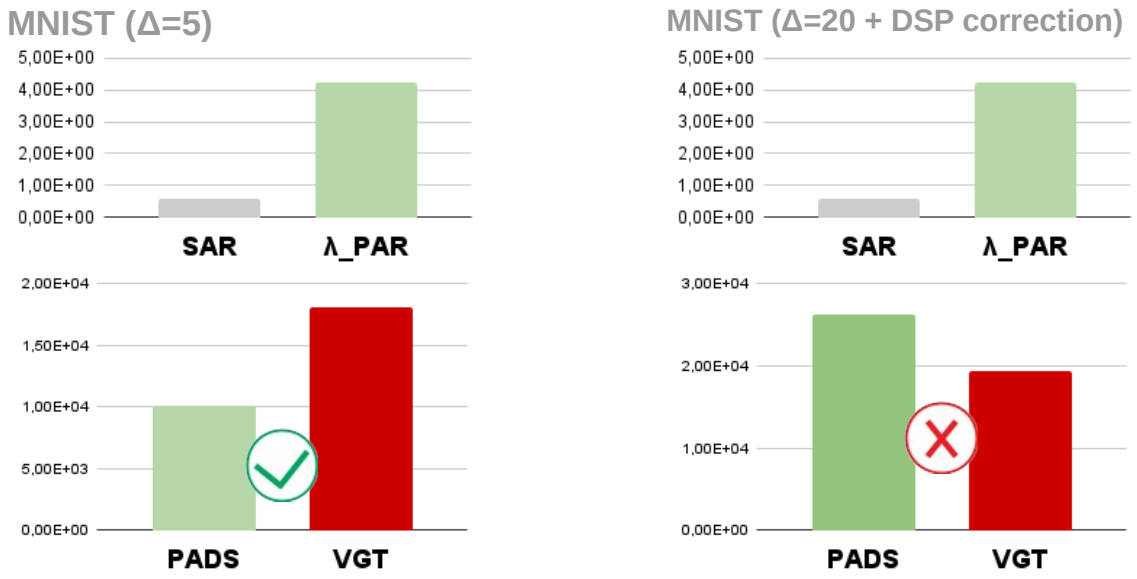
Figure 5.20: SAR and energy estimations without static power consumption, on MNIST ($\Delta = 20$) and OPSSAT($\Delta = 4$)

Lastly, it should be noted that all results were obtained with $\Delta = 20$. That is to ensure the best accuracy, at the expanse of synaptic activity. Indeed the goal of this section is not to provide an energy benchmark for SNN and FNN but rather to validate the SAR model. In Section 4.2, we have shown that SAR increases with Δ. This parameter could be fine-tuned to obtain the best compromise between accuracy and SAR. For example the energy estimation fro VGT and PADS on MNIST with $\Delta = 5$ is provided in Figure 5.21a. In this case, the SAR model is valid, and PADS consumes fewer energy than VGT. The counterpart is a loss of SNN accuracy (98.74% to 97.9%). This observation confirms that tuning Δ leverages the trade-off between energy consumption and performance in rate-coded hardware SNNs. Moreover, the figure shows that PADS provides substantial energy saving over VGT in Mines Versus Rocks. Our intuition is that the small size of input (*i.e.* small timestep length in the spike generation process) enables inference time savings and thus energy savings.

5.7 Conclusion

In this chapter, a framework for hardware-footprint estimation of neuromorphic accelerators was provided. The framework covers wide range of layer sizes for 4 different architectures: PADS and SPLEAT for spiking domain, VGT and C-HLS for formal domain. SPLEAT and C-HLS are highly sequential while VGT and PADS are highly parallel. The framework is based on an hardware measurement campaign, design space interpolation and estimation of inference time and power. In doing so, the framework drastically facilitates the cartography of applications, neural coding domain and level of parallelism. Indeed, the results presented in this section would have required hundreds of hours of synthesis and simulation. Thanks to the framework, such results are obtained in a few minutes.

The framework was applied to Spoken Digits and OPSSAT datasets. Energy is the most commonly limiting factor in embedded systems, thus it is the main metric used for cartography. In this context, the suitability of spiking domain is not straightforward. SPLEAT provided substantial



(a) (b)

Figure 5.21: SAR and energy estimations for MNIST with $\Delta = 5$ (left) and MNIST with $\Delta = 20$ after applying the DSP saturation correction (right)

energy savings over C-HLS (up to a factor 1000), but the architectures have different low-level implementations. Therefore, the generalization of conclusions is mitigated. On the other hand, PADS and VGT have very similar implementations and the comparison is more reliable. In such conditions, spiking domain is responsible for increasing energy consumption by a factor 1.3 on OPSSAT and 3 on Spoken Digits. However, further experiments have shown that PADS could bring energy savings for very low SAR, like on MNIST with $\Delta = 5$. However, the current SAR-based cartography under-estimates PADS energy consumption and is inconsistent with energy estimations. Methods to improve the reliability of the SAR-based cartography will be discussed in the outlooks of this chapter. Moreover, PADS reduces energy consumption by a factor 3.3 in Mines Versus Rocks, despite the SAR predictions. That is thanks to the small size of Mines Versus Rocks samples, which reduces the timestep of the spike generation process in turns reducing inference time and energy consumption. Hence, spiking domain is also adapted to datasets with small samples. In all, the energy overhead of PADS is due to the temporal sparsity of spikes in the current spike encoding. Reducing the temporal sparsity of spikes is thus crucial in order to compete with formal architectures regarding inference time and energy consumption.

However, there are some applications where energy is not the only limiting factor. In drones and satellite applications for example, size is crucial. In solar-powered systems on the other hand, power consumption is the most limiting aspect. Spiking applications could therefore be suited to such applications, as it enables reliable resource and power savings over formal domain. That is, PADS reduces the LUT usage by 50% and DSP usage by 100%, while providing 50% savings in dynamic power. In applications such as on-board satellite cloud segmentation use-case [93] (Section 6.2.2) or object-tracking in drones [108], spiking domain should be preferred over formal inference.

Additionally, the comparison of SPLEAT and PADS yields that high level of parallelism does not always provide speed-up in spiking domain. That is, for low SAR, PADS did not reduce inference time over SPLEAT, both at layer and network level. In such conditions, using lower level

Table 5.1: Dynamic power consumption in PADS with and without power gating, on the benchmark of datasets. VGT dynamic power usage is shown for comparison. Results are shown for the classification stage (FC layers) only. The results have been obtained through the estimation framework, for $\Delta = 20$ (best accuracy).

Classification Stage	**Dynamic Power (mW)**		
	PADS No PG	**PADS PG**	**VGT**
OPSSAT	0.07	0.03	0.15
MNIST	0.97	0.57	2.07
GTSRB	4.04	2.07	12.4
CIFAR-10	0.08	0.02	0.19
MVR	1.07	1.05	1.26
Spoken Digits	1.53	0.98	3.32

of parallelism brings drastic resource and power savings (up to 90%) without increasing inference time. Furthermore, PADS and SPLEAT are both prototypes with extreme level of parallelism. An accomplished accelerator should involve an intermediate level of parallelism, possibly different from one layer to the other. In this context, SAR can be used to tune parallelism at both layer and network level. Such possibility will be investigated further in the outlooks.

5.7.0.0.1 Discussions on power usage In this chapter, we have seen that the power savings provided by spiking domain could be penalized in two ways:

- First, a large part of the dynamic power usage in FPGA systems is due to clock signals. The clocks still use power when the design is in idle state, *i.e.* waiting input spikes. Hence, idle states and active states are equivalent in terms of dynamic power consumption. The Neural Processing Units of PADS are often in such idle state due to the high temporal sparsity of spike encoding. That is the intrinsic sparsity of rate-encoding coupled with the sequential Spike Generation Cell. Therefore, the dynamic power savings brought by the simpler computation are over-compensated by the consumption of idle state.

- Second, power consumption has two components: dynamic power usage mentioned above, and static power consumption. If the dynamic power usage is related to the intrinsic signal toggles in the design, the static power can be considered as an offset depending on the device. For FPGAs, the static component stands for a large part of the overall power consumptions (approximately 25% in a Zedboard at 100% occupation, in average). In turns, the power savings are often not significant in front of the static power offset for FPGA targets. That is even more true with small designs that occupies small portions of the programmable logic. In such conditions, the dynamic power savings are not visible at system level.

In order to simulate the the effect of a strong power gating on spiking architecture, idle power consumption is set to 0 in our estimation framework. This approach neglects the cost of the additional hardware required to manage clock-gating, but it still provides a good estimation of the potential gains. The results on a few datasets are shown in Table 5.1 for information. Hence, this preliminary experiments show that spiking accelerator could indeed mitigate the temporal sparsity of spikes through a power-gated implementations. Such feature should be studied in further work.

On the other hand, a good solution to enhance the power savings of neuromorphic circuits is to address target with lower static-to-dynamic power consumption ratio. For example, ASIC technology demonstrates much lower static power consumptions than FPGAs for equivalent designs. This technology was not available during the PhD thesis, but it should be addressed in further work, possibly at simulation level. Moreover, addressing ASICs instead of FPGA is a much easier task for our estimation approach. If FPGAs have limited resources which creates side-effect during synthesis, this is not so much the case with ASICs. Therefore, such devices seem better suited for our estimation framework approach and this possibility should be studied further.

In all, addressing both of those issues to enhance the power savings of spiking accelerators could in turn leverage the expected energy savings.

5.8 Outlooks

In this section, we provide some outlooks on the future improvement of the work presented in this section.

5.8.1 Improvement of the SAR metric

As shown in the previous section, the SAR metric is not always able to predict which coding domain is preferable in terms of energy consumption. In highly-parallel accelerators (PADS and VGT) that is because the SAR metric does not take spike sparsity into account. To cope with this issue, the SAR metric should be improved by multiplying the result by a variable representing spike sparsity as described in Equation 5.5.

$$\frac{E_{SNN}}{E_{FNN}} = (\frac{SAR}{\lambda}) \times \gamma \tag{5.5}$$

Where γ is the sparsity factor. A possibility would be to set γ to the average time between two consecutive input spikes. Such model should be tested and validated experimentally in further work.

5.8.2 Improvement of the estimation framework

Other sources of error lies in the energy estimation framework. First, the saturation of DSP is taken in account in the SAR model, but not in the network-wise estimations. Indeed, the sum of layer-wise DSP usage often overpasses the limit of the synthesis target (Zedboard). This could cause inconsistency between the SAR model and energy estimations. To cope with this issue, the saturation of DSPs can be approximated using the Table 4.4 of Section 4.3.2. Indeed, this table shows the hardware cost (resource, power and energy) for a MAC acceleration implemented with and without DSP. Hence, the exceeding DSPs can be converted in LUTs and Registers, and the energy consumption corrected accordingly. Such work has been done on MNIST ($\Delta = 20$) as shown in Figure 5.21b. The correction does reduce the error margin, but the SAR-based estimation remains incorrect.

Another way to cope with the issue could be to use ASIC estimations instead of FPGA. Such designs are not affected by such resource saturation, thus no correction is required. Moreover, the FPGA synthesis tool (Vivado) used to obtain all results performs low-level optimizations depending on the board limitations. Such mechanisms could affect the consistency and generalization of the

estimations. Targeting ASIC rather than FPGA could also avoid such concerns. Additionally, the static power consumption in FPGAs is high and often masks the energy savings provided by spiking domain. In ASICs, the static component is much smaller in front of dynamic power usage. As power savings are made on the dynamic component, ASIC technology could highlight the energy benefits of spiking domain. The estimator will be adapted to ASIC technology in further work.

Additionally, C-HLS should be replaced by an architecture closer to SPLEAT in terms of low-level implementation choices. For example, a more relevant formal sequential architecture should have the same inter-layer pipeline as SPLEAT, and the same intra-layer level of parallelism. In doing so, the comparison of SAR predictions and energy estimations will gain relevance.

5.8.3 Studying the level of parallelism

Lastly, this work demonstrated that PADS did not reduce latency compared to SPLEAT when SAR is low. That is, such high level of parallelism is not necessary to sustain the actual density of computation. SPLEAT and PADS are two prototypes which feature two opposed extreme levels of parallelism, but a realistic architecture must involve an intermediate level of parallelism. The goal is to find the lowest possible level of parallelism that does not affect inference time. For example, by using several SPLEAT Neural Processing Unit per layer. In doing so, the quantity of allocated resources and power is strictly adapted to the density of spikes. In further work, an analytic model should be constructed to study this optimal point. SPLEAT will also be adapted to feature intermediate level of parallelism.

5.8.4 Hybridization and other spike encoding methods

This study also brought several insights and perspectives for the remaining of this work. First, coding domain should be tailored to SAR at layer level. For example in Spoken Digits, spiking domain is preferable in FC1 but not in FC2 (in terms of energy consumption). In the light of those observations, using spiking domain on one layer and formal domain on the other could bring the best from both worlds to a same topology. In the next chapter of this thesis (Chapter 6), such concept will be investigated and applied to OPS-SAT and Spoken Digits dataset.

Moreover, temporal sparsity of spikes in the current encoding technique is responsible for the energy overhead in the studied neuromorphic implementations. Hence, reducing temporal sparsity is a crucial issue that must be tackled. Two possibilities arises. First, the temporal sparsity depends on the timestep length in the Spike Generation Cell. This timestep is proportional to the input size. In Chapter 3, down-sampling the input resulted in a loss of information that increased inference time and lowered accuracy. Hybridization could cope with this issue, as spike encoding can be made on intermediate activation (*i.e.* between two layers rather than at the input). Indeed, the size of deep layers are often lower than the size of samples. Second, the temporal sparsity is also intrinsic to rate-based SNNs, as information is encoded through time. In Chapter 7, we investigate a novel type of SNN with lower temporal sparsity and constrained number of timesteps. In the current rate-based SNN model, inference time is not deterministic and drastically vary from one image to the other (up to a factor ten). This could cause timing violation issues for integration in a system. The constrained number of timesteps used for spike encoding in Chapter 7 solves this issue that was let aside in our previous discussions.

Chapter 6

Neural coding domain hybridization

Chapter contents

The idea of tailoring the coding domain at layer level has been mentioned several times in this document. Rather than using a fully-formal or fully-spiking accelerator, the inter-layer variability of SAR and hardware footprint estimation suggests that hybridization could bring benefits and mitigate drawbacks from both worlds. Therefore we propose a novel approach for neuromorphic systems: hybrid acceleration. *i.e.* a combination of formal and spiking accelerators. In this chapter, we focus on a specific type of hybridization:

- Formal feature extraction stage (Convolution & Pooling) for example with VGT,
- Transcoding of feature-maps achieved with the Spike Generation Cell of PADS 3.
- Spiking classification stage (Fully-connected) for example using SPLEAT or PADS Neural Processing Units

First, we introduce the motivations that led to the development of neural coding hybridization. This concept is applied to OPS-SAT and Spoken Digit datasets. We use the estimation framework described in Chapter 5 to estimate energy consumptions on VGT-PADS and VGT-SPLEAT hybrid architectures. For comparison, results for VGT and SPLEAT architectures are also provided. In doing so, the aim of hybridization is to benefit from the low power and logic usage of spiking accelerators, while mitigating their intrinsic time overhead. Third and last, a prototype of VGT-PADS hybrid accelerator for OPS-SAT cloud segmentation task is proposed. This architecture is embedded on-board OPS-SAT satellite for in-flight testing. The hardware measurement, inference time and power usages obtained in real conditions are confronted to the estimations.

6.1 Motivations

In this section, we provide a list of elements which motivated the hybridization approach. Each of the following subsections is dedicated to one particular motivation.

6.1.1 SAR and footprint variability

Results of Chapters 4 and 5 have shown that SAR and energy estimations varied drastically from one layer to another within a same spiking-CNN. As a result, some layers of a model might be suited to spiking acceleration while others are not. This is the case in Spoken Digits ($\Delta = 20$) as reminded in Figure 6.3, where only fully-connected layers have SAR below λ. As explained previously, λ is a value specific to the target (Zedboard) and design which represents the ratio of energy consumption between a MAC and an ACC operation (see 4.3.2). Using spiking coding domain in the classification stage (FC layers) of Spoken Digits might thus bring energy savings locally in the architecture.

6.1.1.1 Input size and loss of information

The influence of the spatial resolution of input data was measured in Section 3.2.3. As explained in Chapter 3 (Section 3.1.1), the temporal sparsity of spike encoding depends on input size. More precisely, the time increment of the spike encoding process (timestep) is equivalent to the sample size in clock cycles. For OPS-SAT, that is $28 \times 28 \times 3 = 2352$ clock cycles (23520 ns). Thus, down-sampling input size was expected to increase the input spike rate through decreasing the GenCell timestep. However, the opposite was observed and down-sampling input size resulted in an

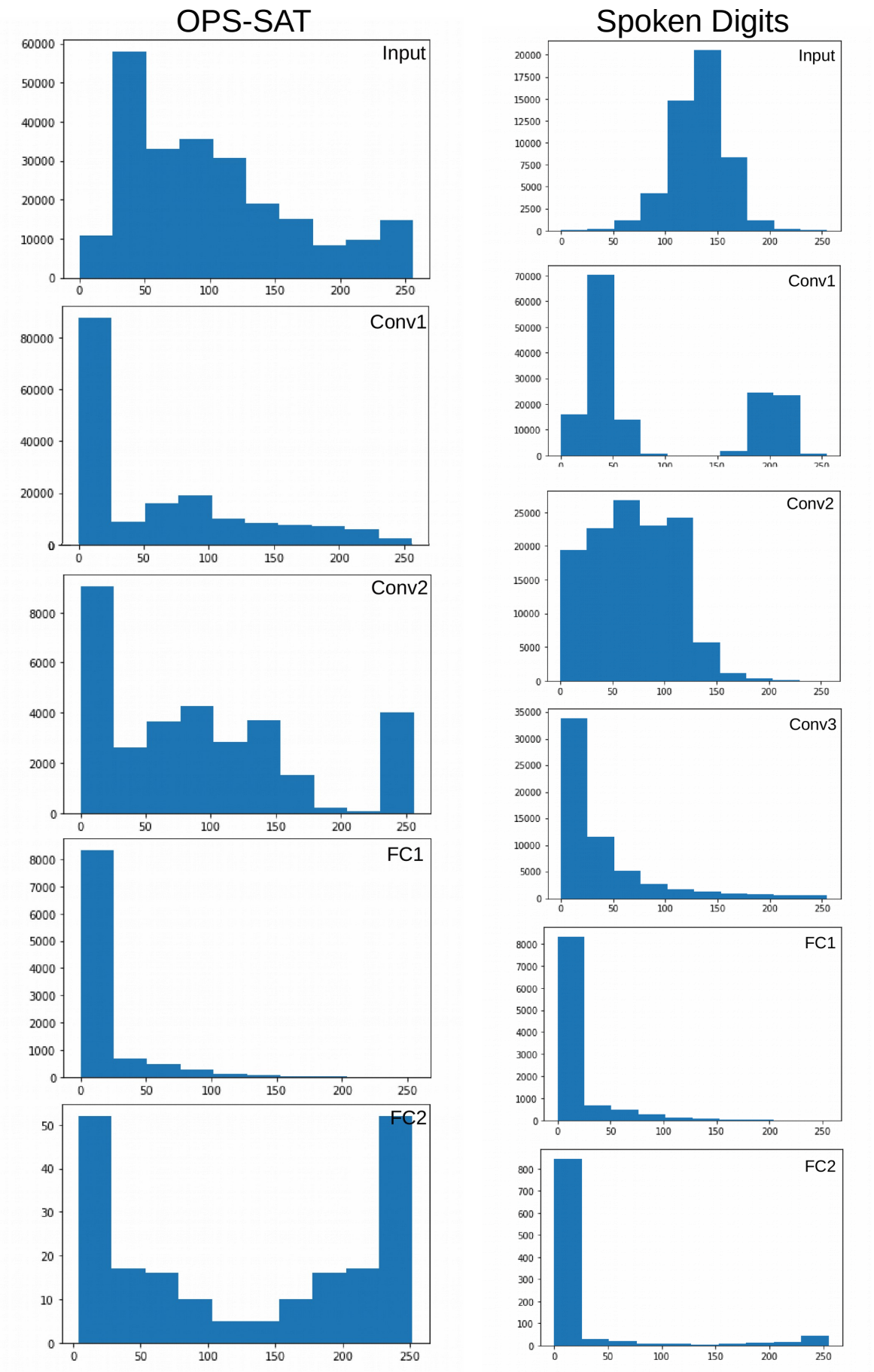


Figure 6.1: Activity distribution in CNN layers for OPSSAT (left) and Spoken Digits (right)

Table 6.1: CNN topologies for a) OPS-SAT and b) Spoken Digits datasets.

(a)

Layer	Type	# Kernels	Kernel size	Stride	Output Shape
0	Input				(28,28,3)
1	Conv	3	5	1	(24,24,3)
2	MaxPool	3	2	2	(12,12,3)
3	Conv	5	5	1	(8,8,5)
4	MaxPool	5	2	2	(4,4,5)
5	FC				10
6	FC				2

(b)

Layer	Type	# Kernels	Kernel size	Stride	Output Shape
0	Input				(507,1)
1	Conv	3	5	1	(503,3)
2	MaxPool	3	2	2	(252,3)
3	Conv	5	5	1	(248,5)
4	MaxPool	5	2	2	(124,5)
6	Conv	5	5	1	(120,5)
7	MaxPool	5	2	2	(60,5)
8	FC				100
9	FC				10

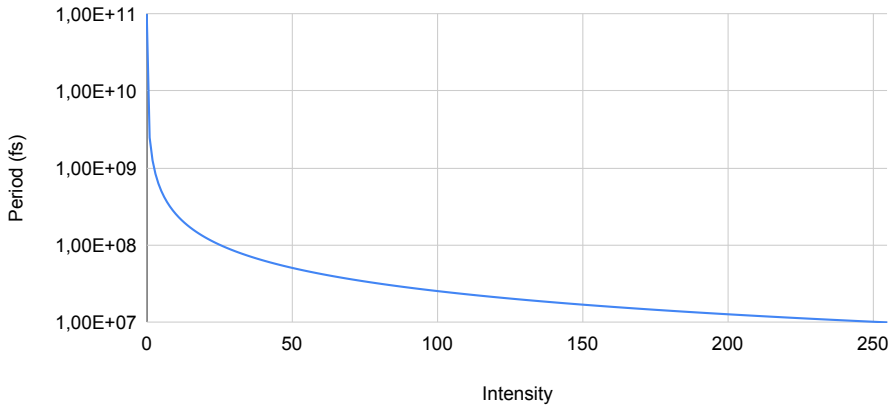


Figure 6.2: Spike train period with respect to element intensity according to the rate-coding policy, with $PeriodMax = 1timestep$ and $PeriodMax = 100timesteps$

extended processing time and lower accuracy. That is because of the arbitrary loss of information caused by down-sampling. The idea behind hybridization is thus to use the first layers of a CNN as a pre-processing for down-sampling without loss of information. Indeed, deeper layer activations often have lower resolution than the input sample. That is the case for FC1 in both OPS-SAT and Spoken Digits CNN topologies 6.1. In doing so, we hope to reduce the spike temporal sparsity and therefore reduce inference time.

6.1.2 Distribution of activity

In Chapter 4 (Section 4.4.3), we have shown that data distribution in vector datasets was not suited to the current rate-encoding policy. As a reminder, the rate-encoding rule is pictured in Figure 6.2: the graph shows that rate-coding policy discriminates pixels with very low intensity from others. Therefore, narrow data distribution such as that of Spoken Digits samples are poorly represented by the rate encoding policy. On the other hand, the distribution of intermediate activation are depicted in Figure 6.1 for each layer of OPS-SAT and Spoken Digits CNNs (Pooling layers have no influence on activity distribution, thus they are not represented). The CNN topologies are described in Table 6.1. In the following, we refer to the output of the feature extraction stage as "latent representation", a common use in Machine Learning. In Figure6.1, the latent representation (Conv2 for OPS-SAT and Conv3 for Spoken Digits) have sparse distributions, with

a lot of pixels with very low intensity. This distribution thus appears to be much more adapted to the rate-encoding policy than the input samples. As explained in section 4.4.3, a better spike-encoding should result in a more efficient processing, *i.e.* less spikes are needed for classification. In turns, this is expected to reduce SAR and in turns inference time and energy consumption of spiking accelerators.

6.1.3 Formal convolutions and spiking classification

Moreover, it should be noted that this work focuses on a specific hybridization scheme where the feature extraction stage operates in formal domain and classification stage in spiking domain. There are a lot of different possibilities of hybridization and our specific approach relies in three elements: First, the number of successive transcoding (passing from one coding domain to another) must not be too great, as encoding and decoding spiking information requires specific hardware which counter-balance the resource and power savings of spiking accelerators. Therefore, we limit our approach to two successive coding domains. Second, a spike encoding module was already developed earlier in the thesis (GenCell), whereas the spike decoding module must be developed from scratch. Indeed, the current Terminate Delta Module only works for categorical classification and cannot provide accurate decoding. Therefore, we concentrate on one single type of transcoding: formal towards spikes. Third and last PADS (which has been developed during this thesis: Chapter 3) only covers spiking fully-connected layers in its current development state, thus it is only able to replace the classification stage for hybridization.

6.2 Estimations on hybrid architectures

In this section, we apply the estimation framework to hybrid architectures on OPS-SAT and Spoken Digits dataset. Such hybrid estimation requires a specific method described in the following.

6.2.1 Methodology

The hybridization approach involves a few differences with the estimations provided in Chapter 5. Indeed, the inference time and dynamic power estimations for the hybrid architectures cannot be made on the original N2D2 activity logs. Indeed, spiking activity is influenced by hybridization, such as a reduction of temporal sparsity as explained in the motivations. This difference must be taken in account as it is one of the major appeal of hybridization: reducing spike encoding temporal sparsity enables to decrease inference time and therefore reduce energy consumption. For each task, the fully-connected stage must be tested in spiking domain separately on a new dataset of latent representation. To do so, we build a dataset of OPS-SAT latent representation and perform spiking inference test using N2D2. The spike encoding is configured with the new *MinPeriod* and *MaxPeriod* values found in Table 6.2. As a reminder, the period values depends on the GenCell timestep length, which itself depends on the sample size: the timestep length is the input size expressed in clock cycles (1clk=10ns at 100MHz). All input sizes and corresponding period hyper-parameters are listed in Table 6.2 for convenience. The timestep used to encode latent representation is indeed much smaller than that required to encode input samples, which promises to reduce encoding temporal sparsity and improve processing speed in the spiking classification stages.

Table 6.2: Input sizes and spike generation min and max period values for full CNN and Hybrid classification stages on OPS-SAT and Spoken Digits dataset.

	OPS-SAT			**Spoken Digits**		
	Input shape (size)	**MinPeriod (ns)**	**MaxPeriod (ns)**	**Input shape (size)**	**MinPeriod (ns)**	**MaxPeriod (ns)**
Full network	28x28x3 (2352)	23 520	2 352 000	507x1x1	5 070	507 000
Hybrid classification stage	80x1x1 (80)	800	80 000	295x1x1 (295)	2 950	295 000

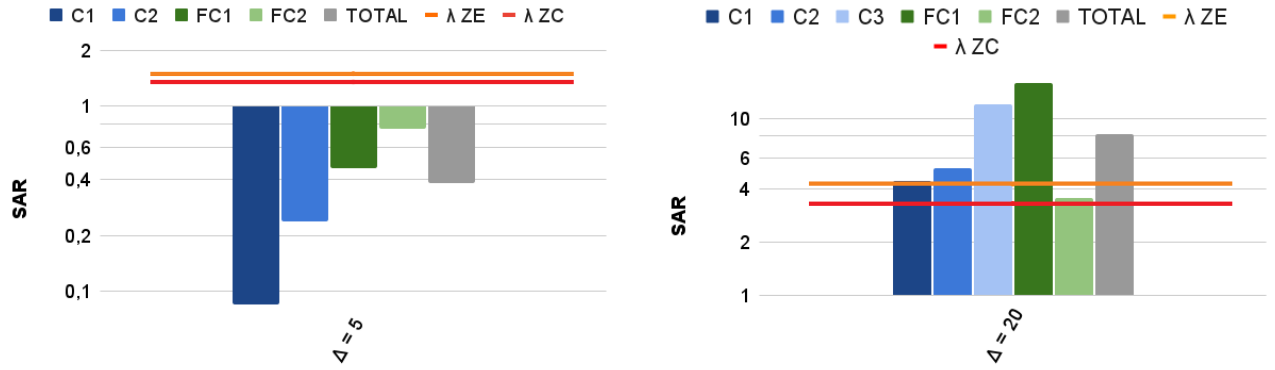
As it is not possible to retrieve the latent representation directly from N2D2, we use TensorFlow instead. To do so, the synaptic weights are exported from N2D2 to TensorFlow, and the intermediates activation are extracted by running the feature-extraction stage and logging the output. Each resulting feature-map is labeled according to the original label of the sample. The classification stage is then tested in spiking domain using N2D2 configured with the periods of Table 6.2. The subsequent synaptic logs are used to estimate time, power and energy in the spiking classification stages of VGT-SPLEAT and VGT-PADS using the estimation framework presented in Chapter 5. For VGT and SPLEAT, the estimations are obtained as described in Chapter 5. It should be noted that there are a few difference between N2D2 and TensorFlow formal inference. Thus, there are slight differences between the estimations provided in this section and that of Chapter 5.

Additionally and in contrast with Chapter 5, we take into account the duration of the GenCell initialization phase in the inference time estimations. This initialization phase lasts for one timestep, as described in Chapter 3. The corresponding time overhead is added to the overall inference time estimations (FE stage for full network and CL stage for hybrid network). Moreover, we also take into account the resource and power usages of the GenCell. To do so, the GenCell are synthesized separately, in the right configurations, to provide accurate resource and power estimations. In future work, those metrics will be directly obtained by incorporating GenCell estimations to the overall framework.

6.2.2 OPS-SAT

In this section, we apply the hardware footprint estimator to OPS-SAT dataset for VGT, SPLEAT, VGT-PADS and VGT-SPLEAT. OPS-SAT dataset seem well suited to hybridization for two main reasons: first, the OPS-SAT feature maps are 30 times smaller than the input samples, which enables to reduce temporal sparsity of spike encoding by a factor 30. Second, the activity distribution is wider in feature-maps than in input samples. Moreover, the OPS-SAT use-case of CIAR project offers to deploy and test the architecture directly on-board a satellite. That is a good opportunity for such a proof of concept. The OPS-SAT use-case will be presented in details in 6.3.1. As a reminder, the CNN used for OPS-SAT classification is available in Table 6.1a.

6.2.2.0.1 SAR on OPS-SAT SAR and accuracy measurements for OPS-SAT spiking CNN are reminded in Figure 6.3. In this section we focus on $\Delta = 4$, as the accuracy improvements achieved for higher Δ values are minimal and are not worth the SAR overhead (see complete Figures in Chapter 4). With $\Delta = 4$, both formal and spiking test reached 81 % accuracy using



(a) (b)

Figure 6.3: Layer-wise and total SAR for a) OPS-SAT ($\Delta = 4$) and b) Spoken Digits ($\Delta = 20$).

N2D2. All layer-wise SAR measurements for OPSSAT $\Delta = 4$ are lower than the λ thresholds, which means all layers are suitable for spiking conversion and not only the classification stage. That is according to the SAR model described in Chapter 4 (Section 4.3.2). As a reminder, λ is the ratio of energy consumption between a MAC operation and an ACC operation. It depends on the hardware target and the number of MACs involved in the formal architecture (VGT). The values have been plotted for Zedboard (λ_{ZE} in orange) and ZCU102 (λ_{ZC} in red). In this chapter, we focus on λ_{ZE} has estimations are made for Zedboard. In Figure 6.3, all layers of the OPS-SAT-related CNN are already suitable for spiking domain, according to the λ model. Therefore, the goal is not to tailor coding domain with respect to SAR, but rather to benefit from the lower input resolution and wide activation distribution (Figure 6.1) of the latent representation. As a reminder, OPS-SAT samples have 2352 values while OPS-SAT latent representation only have 80.

6.2.2.0.2 Estimations on OPS-SAT Before going into further details, we remind the notations used to compare estimations (already used in Chapter 5): $Metric_{Arch}^{Task\text{-}Layer}$ stands for the estimation of a metric on a given dataset, for a given layer (or group of layers) and architecture. For example $E_{SPLEAT}^{Spoken\text{-}Conv}$ is the Energy consumption of SPLEAT Convolution layers on Spoken Digits dataset. Moreover, the Classification stage is noted CL, and the Feature-Extraction stage is noted FE.

The resource, time, power and energy estimations on OPS-SAT dataset are provided in Figure 6.4 for VGT, SPLEAT, hybrid VGT-SPLEAT and VGT-PADS. As already stated in previous chapter, the most limiting factor in most embedded systems is energy. Therefore, we begin by discussing the impact of hybridization in the light of energy consumption. In this chapter, we refer to the Feature Extraction stage (convolution and pooling layers) as FE, and the classification stage (fully-connected) as CL. As shown in the bottom row of the Figure, hybridization enables to drastically reduce the energy overhead of spiking neural networks. For VGT-SPLEAT, on the one hand using the formal VGT FE stage reduces E^{FE} by a factor 11.5. On the other hand, hybridization enables to reduce the GenCell timestep from 2352 clock cycles (*i.e.* sample size) to 80 clock cycles (*i.e.* feature-maps size). This allows to drastically reduce the rate-encoding temporal sparsity, and thus inference time and energy: $E^{\text{CL}}_{\text{SPLEAT}} \approx 20 \times E^{\text{CL}}_{\text{VGT-SPLEAT}}$. This is also visible in the inference time results: $T^{\text{CL}}_{\text{SPLEAT}} \approx 27 \times T^{\text{CL}}_{\text{VGT-SPLEAT}}$. In all, the results show hybridization with SPLEAT enables to reach the same energy consumption than a pure VGT ($E_{\text{VGT}} \approx E_{\text{VGT-SPLEAT}}$).

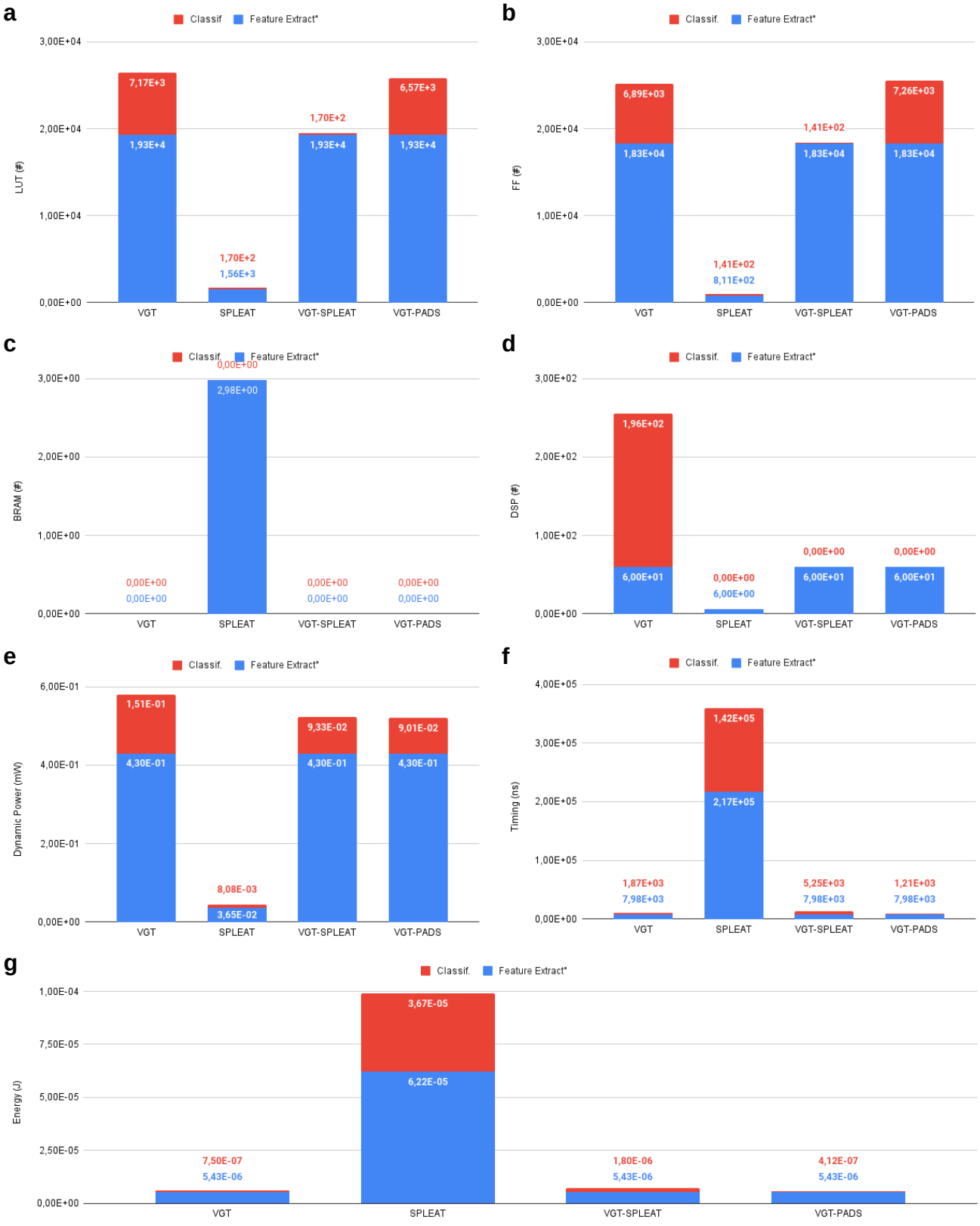


Figure 6.4: Hardware footprint estimation of VGT, SPLEAT, Hybrid VGT-SPLEAT and Hybrid VGT-PADS on OPS-SAT RGB dataset with $\Delta = 4$, $MinPeriod = 1$ and $MaxPeriod = 100$. a) LUT, b) Registers, c) Block RAM, d) DSP, e) Power, f) Inference time and g) Energy.

Similarly to VGT-SPLEAT, hybridization with PADS enables to reduce the energy consumption in the classification stage. Comparing with energy estimations of Chapter 5 for PADS (Figure 5.18) yields that $E^{\text{CL}}_{\text{PADS}} \approx 2.5 \times E^{\text{CL}}_{\text{VGT-PADS}}$. Furthermore, energy estimations without hybridization issued that $E^{\text{CL}}_{\text{PADS}} > E^{\text{CL}}_{\text{SPLEAT}}$. As a reminder, that is because PADS parallelism is not exploited due to spike temporal sparsity of OPS-SAT task: $T^{\text{CL}}_{\text{SPLEAT}} \approx T^{\text{CL}}_{\text{PADS}}$ while $P^{\text{CL}}_{\text{SPLEAT}} < P^{\text{CL}}_{\text{PADS}}$. With hybridization, we observe that $T^{\text{CL}}_{\text{VGT-SPLEAT}} \approx 4 \times T^{\text{CL}}_{\text{VGT-PADS}}$. That is, temporal sparsity of spike is higher and SPLEAT is no longer able to process spike at the same rate than PADS. Consequently, we observe that $E^{\text{CL}}_{\text{VGT-SPLEAT}} \approx 4 \times E^{\text{CL}}_{\text{VGT-PADS}}$. In other words, hybridization increases the parallelism requirements and PADS is therefore preferable in this context. Furthermore, we observe that VGT-PADS CL stages uses even fewer energy than pure VGT CL stage: $E^{\text{CL}}_{\text{VGT}} \approx 2 \times E^{\text{CL}}_{\text{VGT-PADS}}$. That is, hybridization enables to draw energy savings from PADS, whereas it was not the case in Chapter 5. However, at network level $E_{\text{VGT}} \approx E_{\text{VGT-PADS}}$ because most of the energy consumption lies in the VGT FE stage used in both. Therefore, hybridization does not provide significant energy savings on OPS-SAT.

However, using the SPLEAT classification stage instead of VGT enables significant resource savings: the overall LUT and FF usage are reduced by 30% and DSP usage by 80%. Dynamic power usage is also reduced by 10% which is not significant. That is, VGT-SPLEAT hybridization enables to reduce the resource usage of OPS-SAT classification implementation without the usual energy overhead attributed to SPLEAT. For VGT-PADS, the resource savings are much less significant as the OPS-SAT CNN is small: as shown in Chapter 5, PADS resource savings over VGT arise for larger layer sizes. Therefore, we expect more significant resources savings on Spoken Digits Dataset.

In all, estimations show that hybridization offers an interesting trade-off between the resource efficiency of spiking domain and speed of formal accelerators. However, OPS-SAT dataset is not the best application case for hybridization. Indeed, most of computation in VGT on OPS-SAT dataset takes place in the feature-extraction stage. Thus, the transcoding of the classifications stage does not drastically change the overall footprint. In the next section (6.2.3), hybridization is applied to the Spoken Digits CNN, where classification stage is much more important.

6.2.3 Spoken Digits

In this section, we apply the hybridization principle to Spoken Digits datasets whose CNN topology is reminded in Table 6.1. Spoken Digits is well suited to hybridization: the feature-maps are smaller than the input, and show a wide pixel distribution. The narrow intensity distribution of Spoken Digits samples have already been pointed as a cause for the bad SAR scores of spiking CNNs in Section 4.4.3. The much wider distribution of feature-maps should provide a better rate encoding of data. In doing so, we hope to decrease SAR and inference time compared to the original SNN.

6.2.3.0.1 SAR on Spoken Digits The SAR distribution in Spoken Digits is shown in Figure 6.3. In this section, we focus on $\Delta = 20$ to reach the same accuracy as the formal model and provide a fair context for comparisons. As shown in the figure, the FC2 layer is the only one whose SAR is below both λ_{ZE} values for $\Delta = 20$. According to the model, FC2 is thus the only suitable layer for acceleration in spiking domain, whereas the feature extraction stage is not. Thus, in this case the interest of hybridization is not only to benefit from the latent representation small resolution and wide activation distribution, but also to benefit from the low SAR of the FC2 layer.

The hardware estimations for VGT, SPLEAT, VGT-SPLEAT and VGT-PADS are given in Figure 6.5. Like for OPS-SAT, we begin by discussing the energy estimations as this is the most

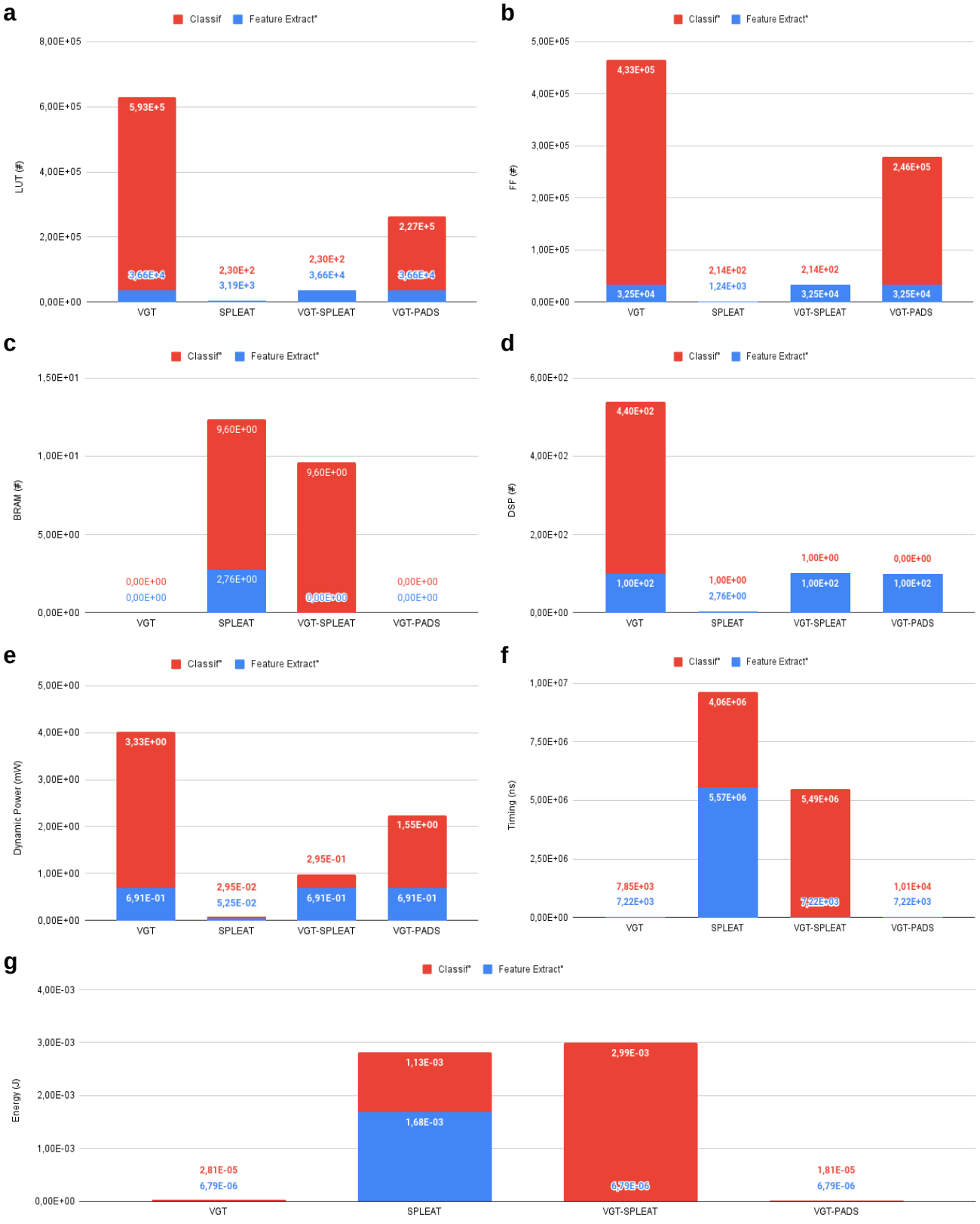


Figure 6.5: Hardware footprint estimation of VGT, SPLEAT, Hybrid VGT-SPLEAT and Hybrid VGT-PADS on Spoken Digits RGB dataset with $\Delta = 4$, $MinPeriod = 1$ and $MaxPeriod = 100$. a) LUT, b) Registers, c) Block RAM, d) DSP, e) Power, f) Inference time and g) Energy.

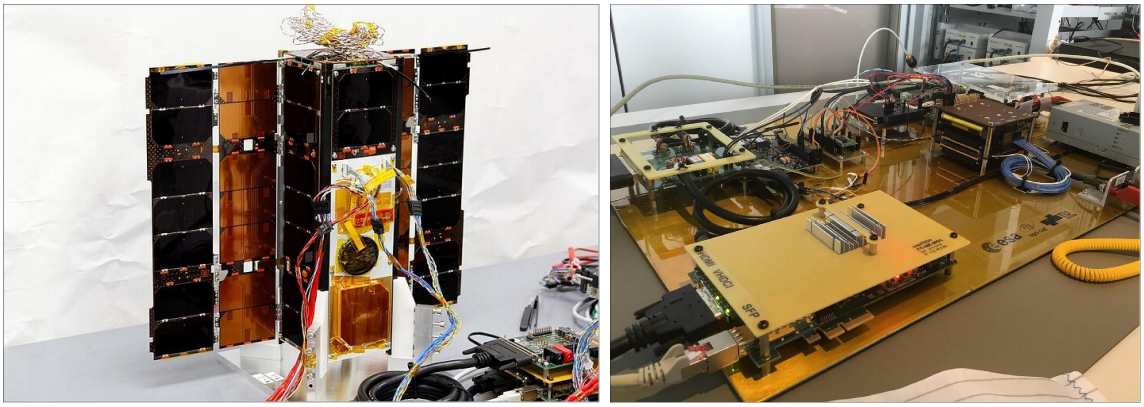
limiting metric in embedded systems. Using the VGT FE stage instead of SPLEAT FE stage enables substantial energy savings: $E^{\text{FE}}_{\text{SPLEAT}} \approx 327 \times E^{\text{FE}}_{\text{VGT-SPLEAT}}$. However, in contrast with OPS-SAT hybridization increases the energy consumption of SPLEAT. Indeed, $E^{\text{CL}}_{\text{VGT-SPLEAT}} \approx 3 \times E^{\text{FE}}_{\text{SPLEAT}}$. As explained in Chapter 5, the density of spikes is so higher in Spoken Digits FC1 ($\text{SAR}^{\text{Spoken-FC1}} \approx 2.8$) that SPLEAT layers are not able to process spikes on-the-fly, even without hybridization. Therefore, reducing temporal sparsity through hybridization does not speed-up SPLEAT inference time. In all, $E^{\text{Spoken}}_{\text{VGT-SPLEAT}}$ is slightly higher than $E^{\text{Spoken}}_{\text{SPLEAT}}$. Furthermore, VGT-SPLEAT is much more energy-intensive than VGT: $E^{\text{Spoken}}_{\text{VGT-SPLEAT}} \approx 86 \times E^{\text{Spoken}}_{\text{VGT}}$ That is, SPLEAT is not suited to hybridization in tasks with high SAR.

The conclusion is very different for VGT-PADS: thanks to the high level of parallelism, PADS is able to process spikes on-the-fly regardless of density. Therefore, reducing temporal sparsity enables to drastically reduce inference time and energy consumption in PADS classification stage. If we refer to the results of Chapter 5, $E^{\text{CL}}_{\text{PADS}} \approx 3600 \times E^{\text{CL}}_{\text{VGT-PADS}}$. Without hybridization, PADS classification stage consumes more energy than VGT (Chapter 5). We observe the opposite with hybridization: $E^{\text{CL}}_{\text{VGT}} \approx 1.6 \times E^{\text{CL}}_{\text{VGT-PADS}}$. In contrast with OPS-SAT, this saving is significant at network level as most of VGT energy consumption lies in the classification stage (80%). Indeed, VGT-PADS provides 30% energy savings over VGT at network level. Therefore, hybridization with PADS provides energy savings where PADS alone does not.

Moreover, energy savings provided by VGT-PADS also come with substantial resource and power savings. Hybridization with PADS enables to reduce LUT usage by a factor 2.4, register usage by 40% and DSP usage by 80%. Moreover, VGT-PADS dynamic power is 45% lower than pure VGT (43% for total power). Those savings are much more significant than on OPS-SAT dataset for two reasons. First, the layers are larger than in OPS-SAT and PADS resource savings arises for larger topologies. Second, most of the resource usage of VGT is located in the classification stage thus savings are significant at network level. In all, VGT-PADS hybridization seems very promising for embedded acceleration, as it enables significant savings in all metrics except inference time. Indeed, VGT-PADS is still 23% slower than pure VGT, but this overhead remains manageable. On the other hand, VGT-SPLEAT also provide substantial resource savings. However it is not necessary to discuss those results as SPLEAT provides even more resource savings for fewer energy.

6.2.4 Discussions on hybrid estimations

In the light of the estimations on OPS-SAT and Spoken Digits dataset, hybridization seems a good opportunity of trade-off between spiking and formal hardware acceleration. Indeed, the classification stage benefits from the resource and power efficiency of spiking domain all the while mitigating the inference time of spiking implementations. This second aspect is achieved by both using a faster formal feature-extraction stage and reducing the GenCell timestep length. Indeed, decreasing the timestep length reduces the temporal sparsity of spike encoding. This enables a drastic speed-up in spiking accelerators provided that they are able to sustain the increased spike density. On OPS-SAT, the SAR is low enough ($\text{SAR} \approx 0.4$). Therefore SPLEAT is able to process spikes on-the-fly, even with a lower spike temporal sparsity. VGT-SPLEAT thus competes with pure VGT in terms of energy consumption on OPS-SAT. However, a higher level of parallelism is required for hybridization on Spoken Digits which features a higher SAR ($\text{SAR} \approx 4$). Under such conditions, spikes arrive faster than the processing ability of VGT-SPLEAT, causing a severe time and energy overhead. On the other hand, VGT PADS is designed to process spikes on-the-fly regardless of the density. Therefore, hybridization with PADS reduces energy consumption by 30%



(a) (b)

Figure 6.6: a: OPS-SAT CubeSat being tested before launch. b: OPS-SAT FlatSat platform. Photo credits: TU Graz [19].

at network-level. On OPS-SAT, the savings are less significant and the energy consumptions are similar at network level. Hybridization also provides substantial resource savings. Like for energy, those savings are more important when the classification stage is larger: VGT-PADS reduces LUT usage by 40%, DSP usage by 80% and total power usage by 43% on Spoken Digits dataset. For the same reason, the savings are less significant on OPS-SAT dataset.

In all, hybridization implies a higher spike density, which requires a higher level of parallelism than a full spiking topology. If the architecture is able to benefit from the lower temporal sparsity of spikes, hybridization provides significant energy savings compared to fully-spiking implementations. Furthermore, hybridization offers significant energy and resource savings over formal implementations, provided that the classification stage is significant in front of the convolution stage. In this work, we only address a simple case where hybridization is limited to the classification stage.

However, hybridization could be made earlier in the network. The trade-off between the speed of formal accelerators and resource and power efficiency of spiking accelerators must be explored further in future works. The SAR metric and estimation framework will enable to find the optimal hybridization scheme for each topology.

In the following section, we propose an hardware implementation for the VGT-PADS hybrid accelerator. The estimations will be confronted to real hardware measurements in order to verify the benefits of hybridization. In doing so, we also provide hardware measurements on VGT and SPLEAT implementations for OPS-SAT dataset. In doing so, we propose to evaluate the reliability of our estimation framework, and strengthen the confidence in the conclusions of Chapter 5.

6.3 Hybrid hardware implementation

In this section, we describe the hardware implementation of the VGT-PADS hybrid neuromorphic accelerator. This accelerator is developed in the context of CIAR project and applied to the OPS-SAT cloud segmentation application. The architecture was embedded onboard OPS-SAT satellite for in-flight testing on real images. In doing so, the system was both the first spiking neural network accelerator in space, and the first hybrid neural network in space. The hardware measurements are confronted with the estimation results.

Figure 6.7: Impression of OPS-SAT in low earth orbit [20]

6.3.1 Context

6.3.1.1 OPS-SAT orbital laboratory

The hybrid neural network accelerator (VGT-PADS) was developed in collaboration with the CIAR project hosted by IRT Saint-Exupéry. The CIAR project addresses deployment of FPGA neural network accelerators in spatial application. More specifically, the CIAR project is implied in experimentation on-board OPS-SAT [109] provided by the European Space Agency (ESA). This satellite is a testing platform for European researchers in the satellite field. It is a CubeSat [110], and measures $10 \times 10 \times 30$cm. It weights 5.5kg. The main advantage of CubeSats is their small size and low weight, which facilitates the access to orbit. Figure 6.6 shows pictures of OPS-SAT and the associated "FlatSat" platform. The "FlatSat" architecture is identical to the satellite and serves as a test platform for applications and experiments. For illustration, an impression of OPS-SAT in low earth orbit is shown in Figure 6.7.

OPS-SAT includes various systems to support a large set of experiments. Each of these modules is specifically hardened for space applications. A description of the full satellite platform is provided in Figure 6.8. However, our experiment only involves the SoC Processor Payload and the HD camera. The camera is used to take photos of the earth to test application in-flight. The OPS-SAT dataset pictures were also taken from this camera, before being manually selected and labeled by CIAR project members. The SoC is a hardened MitySOM, which contains an ARM CPU, DDR4 memory and a Cyclone V FPGA. The VGT-PADS hybrid architecture is developed targeting this board.

6.3.2 VGT-PADS Hybrid Architecture

During CIAR project, the VGT-PADS hybrid hardware architecture was developed aiming in reducing resource and power usage of neural network acceleration for space applications, all the while reducing the intrinsic latency of spiking accelerators. In the present section, we describe the VGT-PADS hardware implementation and confront actual hardware measurements with estimations. Such comparison is made on VGT, SPLEAT and VGT-PADS accelerators. It should be

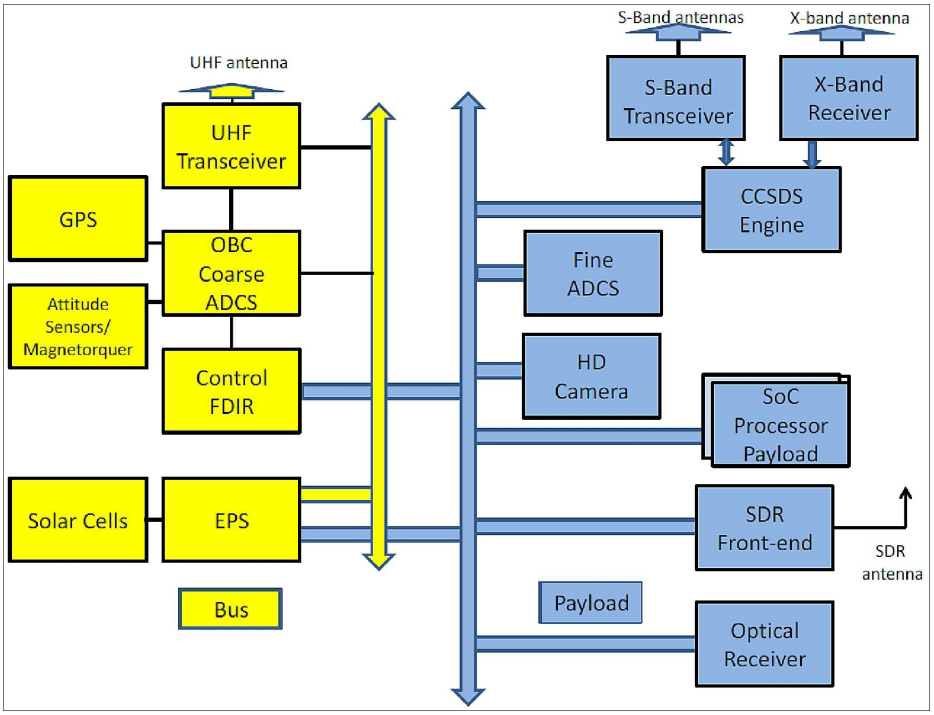


Figure 6.8: Schematic of OPS-SAT architecture. The yellow part is for the "technical" bus, and the blue part is for the payload. Source: [19]

noted that VGT-SPLEAT was not implemented in hardware, due to the aforementioned estimated latency and energy overhead. Therefore it is not featured in this section.

6.3.2.1 Architecture

The VGT-PADS hybrid architecture combines a formal feature extraction stage (conv+pool) with a spiking classification stage (fully-connected). The formal feature extraction stage is implemented using the VGT architecture (Section 2.2.2.1), and the classification stage using PADS FC layers (Section 3.1). The interface between the two is based on the Spike Generation Cell (GenCell). An additional module retrieves the VGT latent representation, performs flattening and streams pixels to the GenCell. A Terminate Delta Module (TDM) is used to decode the spiking output and retrieve the winning class. The architecture is shown in Figure 6.9. The output of VGT feature maps are stored in FIFO queues, one for each channel. When the last pixel is received, the *FM_VALID* signal is raised, triggering the flattening process. Pixels are streamed one by one to the GenCell, following a user-defined flattening policy.

In the following subsection, we provide hardware measurements performed on the VGT-PADS hybrid accelerator, alongside hardware measurements for SPLEAT and VGT. The three accelerators are tested on OPS-SAT dataset. The measurements are confronted to the estimations to confirm the benefits of hybridization and strengthen the confidence in our estimation framework.

6.3.2.2 Results

All three architectures are configured for the OPS-SAT classification tasks, with the synaptic weights resulting from N2D2 training and conversion. Each architecture is synthesized and implemented using Vivado design suite targeting Xilinx Zedboard. The resulting hardware measurement

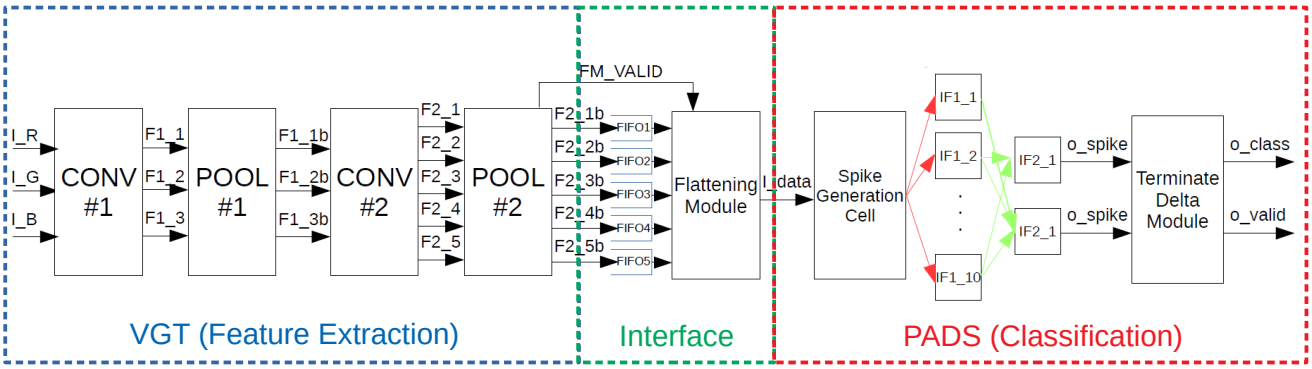


Figure 6.9: Illustration of the CIAR Hybrid Neural Network Architecture.

Table 6.3: Measurements and estimations for VGT, SPLEAT and Hybrid VGT-PADS on OPS-SAT Dataset. The difference is expressed in % of the measurement.

	VGT			VGT-PADS			SPLEAT		
	MEAS.	**EST.**	**DIFF (%)**	**MEAS.**	**EST.**	**DIFF (%)**	**MEAS.**	**EST.**	**DIFF (%)**
Time (ns)	8 835	9 850	11.5	10 985	9 191	16.3	248 073	216 997	13.2
Dynamic Power (W)	0.21	0.58	171.5	0.15	0.52	256.2	0.03	0.04	43.9
Static Power (W)	0.26	0.25	4	0.25	0.25	0	0.25	0.25	0
LUT (#)	22 549	26 500	17.5	17 245	25 900	50.2	2 398	1 730	27.9
FF (#)	30 671	25 200	17.8	23 996	25 600	6.7	724	951	31.3
BRAM (#)	0	0	0	0	0	0	3.5	3	14.3
DSP (#)	760	255	70	200	60	70	8.00	6	25
Energy (nJ)	4 152	8 176	96.9	4 394	7 077	61.1	69 469	62 923	9.4

(resources, power, inference time and energy consumption averaged on 10 samples) are given in Table 6.3 alongside corresponding estimations. The difference between both values is expressed in % of the real measurement. The error margin is displayed in green when it is below 30%, orange when it is around 50% and red when it is greater than 70%. It should be noted that the hardware measurements are different from the original results provided in the CIAR publication [93]. That is because our laboratory uses a different toolchain (Xilinx) than that of CIAR project (Quartus), which imply an incompatibility between the two designs.

Most of the estimations fall within a 15% margin of error with respect to the actual hardware measurement, which is considered acceptable seeing the order of magnitudes involved. However, there appears to be two major sources of errors in our estimations: dynamic power estimation and DSP usage estimation. First, the substantial error for DSP usage estimation can be explained by the difficulty of predicting the low-level Vivado optimizations. Indeed, Vivado is able to multiplex several MAC operations in a single DSP or use LUT and Registers instead. Moreover, the effect of DSP saturation is difficult to extrapolate from layer-level to network-level. However, for both

VGT-PADS and VGT, DSP usage is under-estimated while LUT and Register usage is over-estimated: thus, we assume that Vivado used LUTs and Registers in place of some DSPs in the design. However, the two errors compensate each other (DSPs are used instead of LUTs). Overall, the DSP usage is difficult to estimate due to saturation, multiplexing and alternative synthesis optimizations. To cope with this problem, the hardware database must be rebuilt with no DSP constraints during synthesis. In doing so, we aim in obtaining a flat resource usage estimation that is less influenced by low-level optimizations. Moreover, a source of error arises from the approximation made on VGT convolution layers. In Chapter 5.2.1.1, we explained that the estimations are obtained for 1 input channel, and multiplied by the number of channels. This approximation is valid for resource estimations, but not for power. Indeed, synthesis performs low-level optimization when all channels are synthesized at once. Therefore, the database should be extended to take into account the number of input channels in VGT convolution layers (3D design space).

Second, the substantial error in dynamic power estimation is related to the same kind of issue. Indeed, the network-level dynamic power estimation is computed as the sum of layer-wise estimations. However, Vivado performs low-level optimization such as resource and clock sharing during synthesis. As clocks are responsible for a very significant proportion of dynamic power at layer-level (around 30% in average), clock sharing brings significant dynamic power reduction at network-level. Therefore, the dynamic power estimations demonstrate a wide margin of error compared to actual measurements. In turns, this also affects the energy estimations derived from power and time. However, the conclusions derived from estimations in the previous subsection are still valid, as the comparisons are not too much affected by this error: VGT and VGT-PADS still demonstrate a similar energy consumption, while SPLEAT is much more energy intensive. Therefore, those results strengthen the confidence in the resource and time estimation results, and confirm the benefits of hybridization. However, the dynamic power estimation must be improved to reduce the error margin. This will be addressed in the outlooks section of this Chapter (Section 6.4.1).

6.4 Conclusion

Estimations and hardware measurement results have shown that hybridization represents an interesting trade-off between formal and spiking implementations. It is the most useful in topologies such as Spoken Digits, where the classification stage represents most of VGT energy consumption. On this dataset, VGT-PADS hybrid architecture provides 30% energy savings over pure VGT architecture. It uses 40% fewer LUTs, 80% fewer DSPs and consumes 43% less power. Those substantial savings are promising for the field of embedded artificial intelligence. On OPS-SAT dataset, the classification stage is less significant, thus resource and energy savings are less significant. As explained before, this study focuses on a precise type of architecture, where the feature extraction stage operates in formal domain while the classifier is works with spikes. PADS should be extended to convolution and pooling layers to enable hybridization earlier in the network. The optimal hybridization scheme must be determined for each application in further work, by exploring the trade-off between resource savings and energy savings. More details are given in the outlooks section (Section 6.4.1). Moreover, this study shows that hybridization increases the parallelism and pipeline requirements in the spiking classifier, as spikes arrive more frequently due to the lower temporal sparsity of encoding. This is particularly visible in Spoken Digits results where SAR is high. In such context, SPLEAT demonstrates a severe latency overhead despite the

reduction of temporal sparsity. Therefore, the level of parallelism must be fine-tuned according to the actual temporal sparsity of spikes and SAR in further works. This will be detailed in the outlooks section.

Additionally, the confrontation of estimations on hardware measurements for VGT, VGT-PADS and SPLEAT applied to OPS-SAT dataset strengthen the confidence in the estimation framework, and confirms the benefits of hybridization. Indeed, most metrics where estimated with a margin of error of 15%. Dynamic power estimations suffered severe errors due to unaccounted low-level optimizations performed by Vivado, but all qualitative comparisons remain valid. The improvement of power estimations will be addressed further in the outlooks. Moreover, those results strengthen our confidence in the conclusions drawn from comparisons of formal and spiking accelerators in Chapter 5.

6.4.1 Outlook

In order to improve the estimation framework, we intend to rebuild the hardware footprint database in future work. That is, by using a specific synthesis policy which does not limit the number of available resources. In doing so, interpolation will provide more accurate results, especially for DSP usage. Indeed, such flat resource estimations are less influenced by low-level synthesis optimizations of Vivado. Moreover, we propose to analyze the clock sharing optimizations performed by Vivado when combining layers at network-level. In doing so, we intend in refining the dynamic power estimation and provide more accurate absolute values. Moreover, as explained in Chapter 5, the estimator can be used to determine the optimal level of parallelism in spiking accelerators. This perspective is even more appealing for hybridization where parallelism requirements are higher.

Finally, another way to improve the energy efficiency of hardware SNNs is to addressed new type of spike coding which demonstrate lower temporal sparsity and SAR. In Chapter 7, we will also address such new representation with SNNs trained through Surrogate Gradient Learning (SGL) [32] [73]. In addition to lower SAR, this type of SNN is timestep-constrained, thus inference time does not depend on synaptic activity but rather on a user defined hyper-parameter. Such feature could increase the appeal for hardware SNN accelerators as it facilitates the timing management at system level. The SAR for SG learning will be measured on the dataset benchmark, and PADS will be adapted to this new SNN model.

Chapter 7

Enhancing PADS: FISO & LIF as Recurrent neurons

Chapter contents

In this chapter we address a novel representation of SNNs proposed by Neftci *et. al.* in [32]. This new type of SNNs represents Leaky Integrate and Fire (LIF) neurons as Recurrent Neurons with binary outputs and inputs. Indeed, LIF and Recurrent Neurons share major similarities such as their similar architecture, temporal dynamics and training through weight adjustment. Considering the SNN as an RNN enables to use existing and mature training methods, such as the well-established Backpropagation Through Time (BPTT) algorithm. This algorithm is adapted to the LIF neuron activation function using the Surrogate Gradient Learning (SGL) technique [32] [91]. This method has two main advantages: first SNNs trained through Surrogate Gradient Learning demonstrated lower spiking activity and inference time for comparable accuracy than converted rate-coded SNNs [73]. Moreover, training in spiking domain enables to penalize spiking activity during training. This novel model of SNNs is promising for hardware acceleration. Indeed, the previous chapters of this thesis have shown that minimizing spiking activity (*i.e.* SAR) and inference time is a crucial matter to leverage energy savings in neuromorphic accelerators.

The SNN model proposed in [32] is described in details in Section 7.1. We also describe the S2NET framework proposed by Zimmer *et. al.* in [21]. This pytorch-based training framework implements the Surrogate Gradient Learning technique alongside Send-on-Delta spike encoding. It is used to measure accuracy and SAR on the benchmark of datasets (MNIST, OPSSAT, GTSRB, CIFAR-10, Mines VS Rocks, Spoken Digits and RadioML 2018) in Section 7.2. The results are confronted with that of rate-coded SNNs obtained in Chapter 4. Lastly, we propose a first prototype of hardware implementation of such SGL-trained SNN model. The architecture is based on PADS and adapted to the LIF dynamics of S2NET.

7.1 Theoretical Background

In this section, we describe the SNN model proposed in [21]. Rather than rate-encoding like in previous chapters of this thesis, the authors used the Send-on-Delta [33] technique which tailors temporal sampling to the signal gradient. This encoding scheme is described in Section 7.1.1. Moreover, the model is based on the equivalence between LIF neurons and recurrent neural cells. The LIF neuron dynamics are described in Section 7.1.2, and the equivalence with recurrent neurons is discussed in Section 7.1.2. Thanks to this equivalence, SNNs can be trained using algorithms developed for Recurrent Neural Networks (RNN) such as the Backpropagation Through Time (BPTT) algorithm. In section 7.1.4, we describe the Surrogate Gradient Learning technique proposed in [32]. This algorithm is an adaptation of the BPTT to the LIF activation function. The output spike decoding process is described in Section 7.3.1.4. In Section 7.1.6, we describe the simplification to the model brought by the S2NET framework. Since this framework and the SNN model were originally intended for temporal signal processing, Section 7.1.7 describes how the model is adapted to static samples.

7.1.1 Send on Delta spike encoding

The Send-on-Delta (SoD) spike encoding method is inspired from [33], in which Miskowicz *et. al.* proposed a strategy for information encoding in event-based sensors. This method is adapted in the S2NET framework which is used in this chapter. It is based on a signal-dependent sampling scheme which is able to represent information in fewer spikes (sparse coding) than conventional rate-based methods. According to the authors and to the conclusions of this thesis so far, such a feature could provide substantial energy savings in clock-gated or power-gated hardware SNN

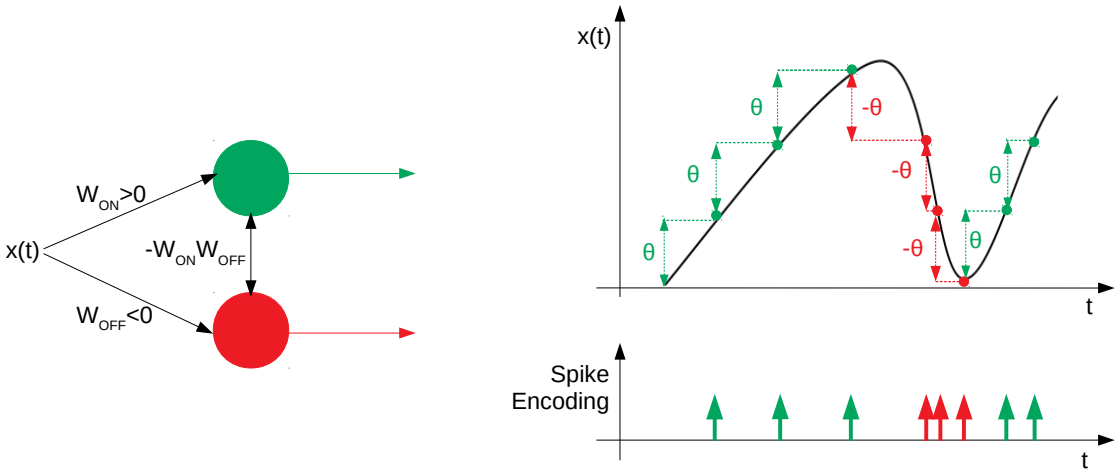


Figure 7.1: Illustration of the Send-on-Delta spike encoding process used in [21], for 2 IF neurons encoding a 1D temporal signal. Left: network setup. Right: input and output data.

implementations (see outlooks of Chapter 5). The SoD method is based on lateral connections between spiking neurons. In Figure 7.1, the SoD method is illustrated for a simplistic case where 2 Integrate & Fire (IF) neurons encode a 1D temporal signal. The goal of SoD is to detect increase or decrease in the input data. The IF neuron is able to detect an increase in input signal through a threshold mechanism and membrane potential reset. In Figure 7.1, the threshold is noted Θ. The green neuron detects increases thanks to a positive synaptic weight whereas the red neuron detects decreases thanks to a negative weight. As shown in the figure, the sampling rate is not arbitrary fixed but depends on the temporal variations. Moreover, the lateral connections enables to reset the membrane potential of both neurons whenever one of them emits a spike.

This mechanism can be generalized to populations of N neurons to process temporal input signals with N dimensions. For example, this mechanism can be applied to layers of Gabor filters [111] which are used to detect edges in 2D images. In this chapter, the SoD encoding is used in all layers of the networks. That is, the first layer of neurons performs spike encoding of analog inputs through SoD, and the following layers of LIF neurons do the same of spiking inputs. Additionally, the parameters of SoD (synaptic weights and neuron threshold) are learned during training. The resulting spike encoding scheme is therefore optimized to minimize spiking activity and maximize accuracy.

7.1.2 LIF Neuron

The LIF neuron is similar to the IF neuron used in the previous chapter of this thesis. The difference lies in the leakage of the membrane potential responsible for the Leaky Integrate & Fire appellation. The dynamics of this neuron illustrated in Figure 7.2 and described in Equation 7.1:

$$\tau_{mem}\frac{dU_i^l(t)}{dt} = -(U_i^l(t) - U_{\text{rest}}) + RI_i^l(t) \tag{7.1}$$

Where U_i^l is the membrane potential of the i^{th} neuron of layer l, τ_{mem} is the membrane time constant, U_{rest} is the resting potential, R the input resistance and I the input current. Whenever the membrane potential overpasses the neuron threshold θ, a spike is emitted in output and the

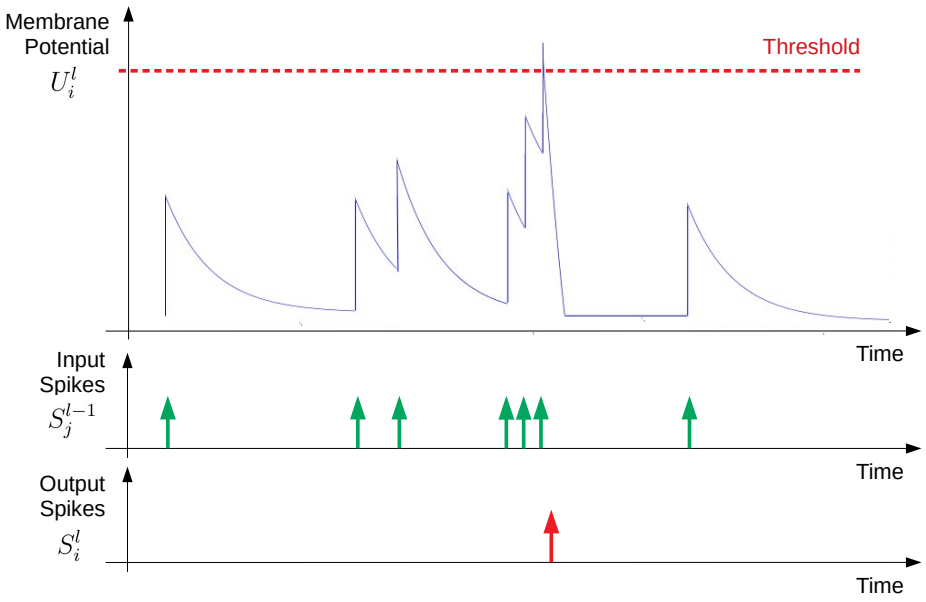


Figure 7.2: Illustration of the Leaky Integrate & Fire neuron process. The membrane potential is shown in blue, input spikes in green and output spikes in red.

membrane potential is instantaneously decreased by $\theta - U_{\text{rest}}$. The output spike train is noted $S_i^l(t) = \Theta(U_i^l(t) - \theta)$ where Θ is the Heaviside step function. Incorporating this threshold term in Equation 7.1 yields:

$$\frac{dU_i^l(t)}{dt} = -\frac{1}{\tau_{\text{mem}}}((U_i^l(t) - U_{\text{rest}}) + RI_i^l(t)) - S_i^l(t)(\theta - U_{\text{rest}}) \tag{7.2}$$

The input current is generated by input spikes coming from neurons of the layer $l-1$ as described in Equation 7.3.

$$I_i^l(t) = \sum_j w_{i,j} \times S_j^{l-1}(t) + \sum_j v_{i,j} \times S_j^l(t) \tag{7.3}$$

Where $\sum_j w_{i,j} \times S_j^{l-1}(t)$ is the feed-forward term (influence of input spikes) and $\sum_j v_{i,j} \times S_j^l(t)$ is the recurrent term for lateral connections (see Section 7.1.1). $w_{i,j}$ and $v_{i,j}$ are the synaptic weights of the input and recurrent synapses respectively. In all, the LIF neuron is able to process temporal information thanks to the membrane potential leakage. Indeed, the relative timing of input spikes has no influence on the IF neuron potential, whereas it does in the LIF neuron. This features also decreases the number of spikes in the networks, as all membranes are constantly decreasing.

7.1.3 LIF as recurrent neurons

The solution of the differential equation 7.1 can be approximated on discrete time through the Euler Method as shown in Equation 7.4. Moreover, we set $U_{rest} = 0$ and $R = 1$.

$$U_i^l[t+1] = e^{\frac{-\Delta t}{\tau_{\text{mem}}}} \times (U_i^l[t] - S_i^l[t] \times \theta) + (1 - e^{\frac{-\Delta t}{\tau_{\text{mem}}}}) \times (\sum_j w_{i,j} \times S_j^{l-1}[t] + \sum_j v_{i,j} \times S_j^l[t]) \tag{7.4}$$

Where Δt is the time increment between two timesteps. According to Neftci *et. al.* [32], those equations precisely describe a Recurrent Neural Network with binary output and whose activation

function is an Heaviside (threshold) function. Thanks to this equivalence, the network of LIF neurons can be trained using the well-established RNN training methods, such as the BPTT algorithm. The BPTT algorithm is an adaptation of the static BP algorithm for dynamic pattern recognition used in formal RNNs. However, the Heaviside activation function is not adapted to the BPTT algorithm. In the next subsection, we describe how BPTT is adapted to networks of LIF neurons through Surrogate Gradient Learning.

7.1.4 Surrogate Gradient Learning

In the BPTT algorithm, the adjustment of synaptic weights is proportional to the partial derivatives of the activation function. Since the activation function of the LIF neuron is an Heaviside step-function, two issues arise:

- First, the Heaviside function is not derivable in 0,
- Second and most importantly, the derivative of the Heaviside function is 0 on $R*$.
- In all, the weight adjustment factor cannot be computed in zero, and it is equal to zero everywhere else.

To cope with these issues, Neftci *et. al.* [32] proposed to use an approximation of the Heaviside function in the backpropagation algorithm: a sigmoïd function. In other word, the activation function is a sigmoïd during the backward path, and an Heaviside during the forward path. Using this technique, several papers in the literature have demonstrated state-of-the-art classification accuracy with reduced synaptic activity [73] [32] [91] [21]. As explained before, training in spiking domain enables to optimize the spiking activity: the number of spikes can be penalized at network-level by taking it into account in the loss function. In the S2NET framework [21], all layers of the networks are trained using the Surrogate Gradient Learning technique. That is, including the first layer which performs SoD encoding of analog input data.

7.1.5 Output decoding: readout layer

Finally, the output spiking activity is decoded through a readout layer. This layer is made of non-firing neurons with no lateral connections. In classification tasks, each class is associated with an output neuron like for Terminate Delta (Chapter 2). The probability of each class is defined by the normalized average membrane potential of the corresponding neuron. This method has been proposed by Zimmer *et. al.* in [21] as a simplification of the model developed by Neftci in [32], which implied a softmax function. According to the authors, the training is more stable with an average function. Moreover, the latter is more hardware-friendly than the complex softmax function.

7.1.6 Application in the S2NET framework

In the next section, we use the S2NET framework proposed by Zimmer *et. al.* in [21], which implements the aforementioned SNN model and Surrogate Gradient Learning in PyTorch. Using this framework, we measure accuracy and SAR on the benchmark of datasets. The results will be confronted to that of rate-coded converted SNNs. In this framework, the neuron threshold (θ) is a trainable parameter which is common to the layer. The neuron threshold is analogous to Δ

Table 7.1: Parameters used for training with S2NET framework

Parameter	Value
Sigma value	5
Learning Rate	0.0005
Weight Decay	0.00004
Reg. Loss Coef.	0
Optimizer	RAdam
Gamma	0.8
Step Size	3
# Epochs	100

in the Send-on-Delta encoding method, but we use θ for the sake of consistency with previous chapters. Moreover, the exponential decay of the membrane potential is approximated by a linear decay (β). This parameter is also trained through Surrogate Gradient Learning, and shared among all neurons of a layer. Finally, in the S2NET framework uses a threshold balancing technique is applied: the neuron threshold θ is normalized by the norm of the synaptic weight matrix. The specific LIF dynamics implemented in the S2NET framework are summarized in Equation 7.5:

$$\begin{aligned} U_i^l[t+1] &= (U_i^l[t] - S_i^l[t] \times \theta^l \times B_i^l) \times \beta^l + (\sum_j w_{i,j} \times S_j^{l-1}[t] + \sum_j v_{i,j} \times S_j^l[t]) \times (1-\beta^l) \\ S_i^l[t] &= \Theta(U_i^l[t] - \theta^l \times B_i^l) \\ B_i^l &= ||w_{i,j}||^2 \end{aligned} \tag{7.5}$$

Where $U_i^l[t]$ is the membrane potential of neuron i of layer l at timestep t, $S_j^{l-1}[t]$ is the state of the input synapse coming from neuron j of layer $l-1$ at time t, $w_{i,j}$ is the synaptic weight between neuron i and j, β^l is the membrane decay factor of layer l, θ^l the threshold of neurons in layer l and Θ is the Heaviside function. The threshold is normalized by B_i^l to balance weights and threshold. It should be noted that $w_{i,j}$, β^l and θ^l are trainable parameters, the two latter being common to all neurons of a same layer.

7.1.7 Static input samples

The SNN model described in this section was originally developed to process temporal data, such as time series, video recordings or data from dynamic vision sensors. In practice, the continuous time is discretized in timesteps like shown in Equations 7.4. Moreover, the model is also applicable to static data like the one used in our benchmark of dataset (Chapter 4). To do so, the same input sample is presented for several successive time-steps. The number of timesteps will define the spike encoding window for all layers of the neural network, *i.e.* If the input sample is presented for five timesteps, spikes will be encoded on five timesteps in all successive layers.

7.2 Accuracy & SAR results

The model described in previous section (Section 7.1) is applied to the benchmark of datasets: MNIST, GTSRB, CIFAR-10, Mines Vs. Rocks, Spoken Digits and RadioML 2018. For each dataset, we measure accuracy and SAR for various number of timesteps. As explained before, the

Table 7.2: Comparison of accuracy and network-wise SAR with Conversion with Rate-coding and SGL with SoD. Bold letters show the best-case accuracy for reach method and task.

	Conversion \& Rate			SGL \& SoD		
Task	**Δ**	**SNN ACC. (%)**	**SAR**	**TS**	**SNN ACC. (%)**	**SAR**
MNIST λ_{ZE}=4.2 / λ_{ZC}=2.7	5	97.3	0.58	5	98.5	0.60
	10	98.9	0.82	**10**	**98.6**	**1.41**
	20	**99.1**	**1.24**	20	98.6	3.12
GTSRB λ_{ZE}=4.4 / λ_{ZC}=5.2	5	83.3	2.24	5	96.7	1.18
	10	90.6	3.59	**10**	**96.8**	**2.33**
	20	**93.5**	**7.70**	20	96.6	4.92
CIFAR-10 λ_{ZE}=4.4 / λ_{ZC}=5.4	5	31.8	0.94	5	53.4	0.82
	10	50.2	2.57	**10**	**53.7**	**1.69**
	20	**57.8**	**7.57**	20	53.7	3.58
Mines V Rocks λ_{ZE}=4.0 / λ_{ZC}=1.5	5	74.2	0.58	**5**	**83.8**	**1.48**
	10	75	1.07	10	83.3	3.11
	20	**78.8**	**3.34**	20	75.7	6.42
Spoken Digits λ_{ZE}=4.4 / λ_{ZC}=5.4	5	38.2	2.04	5	79.7	1.24
	10	71.6	6.41	10	80.7	2.94
	20	**87.5**	**8.24**	**20**	**81.5**	**6.49**
RadioML 2018 λ_{ZE}=4.4 / λ_{ZC}=5.7	5	20.5	6.20	5	58.4	0.98
	10	31.6	11.45	10	60.0	2.21
	20	**37.5**	**17.41**	**20**	**60.7**	**4.85**

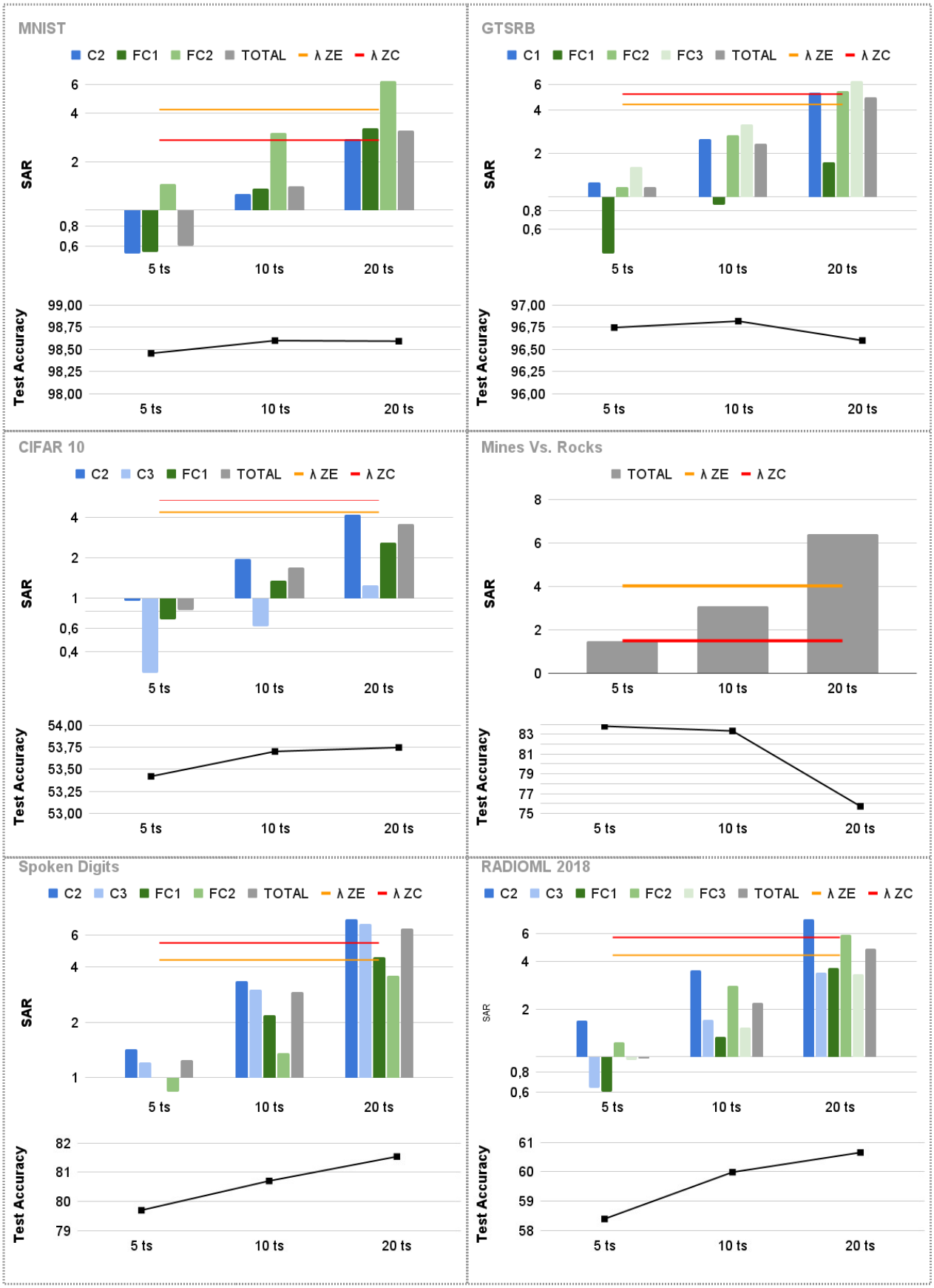


Figure 7.3: SAR and Accuracy measurements on the benchmark of datasets with S2NET framework. Samples are presented for 5, 10 and 20 time-steps. The λ value is depicted in orange for the Zedboard and red for the ZCU102 (see Section 4.3.2). All measurements are averaged on 10 runs.

static samples are presented for a given number of time-steps. This number defines the width of the spike-encoding window. In other words, if the sample is presented for three timesteps, spikes are encoded on three timesteps in all the network. Since a larger time-window enables to encode richer information using more spikes, we suspect a trade-off between SAR and accuracy which can be leveraged by tuning the number of timesteps. The results of this set of experiments is shown in Figure 7.3. All values are averaged on 10 runs. The same parameters listed in 7.1 are used for all tasks to enhance the intrinsic influence of the data. It should be noted that OPS-SAT is not covered in this study due to technical issues.

We begin by comparing the network-wise SAR and accuracy obtained with S2NET with the previous results obtained with Conversion and rate-encoding using N2D2 (Chapter 4). In order to facilitate the discussion, the network-level results are summarized in Table 7.2. As a reminder, the SAR is considered suitable for spiking acceleration when it is below the λ value, according to the model proposed in Section 4.3.2. λ represents the ratio of energy consumption between a MAC operation and an ACC operation on FPGA. It depends on the target board and on the baseline formal design (VGT formal parallel accelerator). In Table 7.2, we provide λ_{ZC} for the ZCU102 board, and λ_{ZE} for the Zedboard. The SAR is displayed in green if it is below both λ values, orange between and red over. Moreover, the best-case accuracy is displayed in bold letters to enhance comparison between the two training methods. According to the results, Surrogate Gradient Learning does not always provide better accuracy than conversion. The latter indeed performs better on MNIST, CIFAR-10 and Spoken Digits datasets. However, the same default training parameters (Table 7.1) were used for all S2NET training runs. That is to enhance the comparison between datasets, but a fine tuning could provide better classification accuracy in further works. Indeed, Surrogate Gradient Learning is able to reach state-of-the-art accuracy according to the literature. On the other hand, SGL performs significantly better than conversion on on GTSRB and RadioML 2018 datasets. On the latter, the best-accuracy is nearly two times higher with Surrogate Gradient Learning.

However, the main matter of this study is not accuracy but Synaptic Activity Ratio (SAR). According to the results, SGL achieves substantially lower SAR than conversion for most datasets. That is apart from MNIST, in which SAR are very close for both methods. As depicted in Table 7.2, the best-case accuracy for conversion is reached for SAR deemed unsuitable for spiking acceleration (red). Using SGL, not only a better accuracy is reached, but the SAR is also suitable for spiking acceleration (green). In Spoken Digits, the best case accuracy with SGL is associated to a red SAR. However, the SAR becomes favorable to spikes if the number of timesteps is reduced from 20 to 10. Under such conditions, the model only looses 1% accuracy compared to the best-case. For Conversion, the only configuration suitable for hardware acceleration is $\Delta = 5$, but the accuracy is only 38.2%. In all, Surrogate Gradient Learning and Send-on-Delta encoding are more adapted to hardware acceleration than conversion and rate-coding. Furthermore, increasing the number of timesteps does not always provide a better accuracy. For MNIST, CIFAR-10 and GTSRB, increasing the number of timesteps over 10 degrades accuracy. Therefore, increasing the number of timesteps arbitrarily is counter-productive and each model should be tuned experimentally.

Additionally, the reduction of synaptic activity is not the only advantage of the SNN model proposed in the S2NET framework. As explained before, the spikes are encoded on a time-window with fixed length. This is a major difference with rate encoding in which the number of timesteps is non deterministic and depends on spiking activity and Terminate Delta termination condition. For hardware acceleration in highly parallel architectures, this means that the inference time is fully deterministic. The integration of such hardware accelerator in wider systems is therefore

facilitated by the easier management of timings. Moreover, the timestep constraint of this SNN model enables to tune the width of the spike encoding window. On the other hand, the window width is data dependent in rate-coding: it cannot be tuned and can grow to very large numbers in some cases. This matter will be studied at the end of the next section.

7.3 PADS V2

In this section, we propose an hardware architecture dedicated to the acceleration of SNN inference, using the model described in the previous section (7.1). This architecture is based on PADS: it is fully parallel and pipelined. Moreover, only fully-connected layers are supported. In the following, the previous version of PADS is referred to as PADS-V1, and the new iteration as PADS-V2. The architecture is described in details in Section 7.3.1. Then, we confront PADS-V1 and PADS-V2 in terms of inference time on a few examples. The aim of this study is to determine if this SNN model used in recent frameworks such as S2NET [21] could provide lower inference time than rate-coded converted SNNs in FPGA accelerators.

7.3.1 Architecture

The architecture is made of three distinct modules, each of which is based on the Generic Neural Processing Unit of PADS-V1. Those modules are:

- The Formal Input Spiking Output (FISO) layer to encode pixels into spike trains,
- The LIF layer made of LIF neurons for hidden layers
- The Readout layer made of non-firing neurons and a dedicated module to compute the maximum membrane potential value.

The architecture is fully-parallel, meaning that each neuron is physically implemented in hardware like in PADS-V1. Moreover, the architecture features an end-to-end pipe-line from spike-encoding (FISO layer) to spike-decoding (Readout layer) to reduce inference time. After describing each module in a dedicated subsection below, the last paragraph describes the mechanism of pipe-line synchronization implemented in PADS-V2. It should be noted that this feature was note implemented in PADS-V1. Before going into further details, we provide some simplifications of the model in the first subsection.

7.3.1.1 Simplification

First, the lateral connections are not implemented in the current stage of development. However, the S2NET network enables to deactivate lateral connection for training and testing. Without lateral connections, the accuracy is slightly degraded and the number of spikes is higher, but the model is still functional. This feature is used to validate the architecture in it current state of development. Moreover, the membrane potential equations of S2NET are expressed differently to facilitate hardware implementation. The goal of those simplifications is to enable as much off-line computation as possible. In doing so, we avoid costly hardware operations such as multiplication and divisions. As a reminder, the original model proposed in [21] (without lateral connections) is

shown in Equations 7.6:

$$\begin{aligned} U_i^l[t+1] &= (U_i^l[t] - S_i^l[t] \times \theta^l \times B_i^l) \times \beta^l + (\sum_j w_{i,j} \times S_j^{l-1}[t]) \times (1-\beta^l) \\ S_i^l[t] &= \Theta(U_i^l[t] - \theta^l \times B_i^l) \\ B_i^l &= ||w_{i,j}||^2 \end{aligned} \tag{7.6}$$

Where $U_i^l[t]$ is the membrane potential of neuron i of layer l at timestep t, $S_j^{l-1}[t]$ is the state of the input synapse coming from neuron j of layer $l-1$ at time t, $w_{i,j}$ is the synaptic weight between neuron i and j, β^l is the membrane decay factor of layer l, θ^l the threshold of neurons in layer l and Θ is the Heaviside function. The threshold is normalized by B_i^l to balance weights and threshold. It should be noted that $w_{i,j}$, β^l and θ^l are trainable parameters, the two latter being common to all neurons of a same layer. The equation of the membrane potential can be expressed in a different manner:

$$\begin{aligned} U_i^l[t+1] &= U_i^l[t] \times \beta^l - S_i^l[t] \times \theta^l \times B_i^l \times \beta^l + (\sum_j w_{i,j} \times (1-\beta^l) \times S_j^{l-1}[t]) \\ S_i^l[t] &= \Theta(U_i^l[t] - \theta^l \times B_i^l) \\ B_i^l &= ||w_{i,j}||^2 \end{aligned} \tag{7.7}$$

In this equation, a lot of multiplications can be made off-line. That is the case for the following products: $\theta^l \times B_i^l \times \beta^l$, $w_{i,j} \times (1-\beta^l)$, $\theta^l \times B_i^l$ and $||w_{i,j}||^2$. Those values are therefore pre-processed and either stored in RAMS (synaptic weights $w_{i,j} \times (1-\beta^l)$) or hard-coded in the neuron. Deporting those operations off-line enables to relieve the number of on-line operations supported by the programmable logic.

Moreover, the Readout layer process is also simplified. In the S2NET model, the Readout layer is made of non-firing leaky neurons. At the end of processing, the winning neuron is selected as the one whose average membrane potential is the highest. It should be noted that this is a temporal average across the discrete time window. However, the goal of the Readout layer is mostly to determine the most active output neuron. The same thing can be achieved by using non-leaky in addition to non-firing neuron, and finding the maximum total membrane potential rather than the maximum average. In doing so, the architecture is relieved from costly multiplication (leakage) and division (average) operations.

7.3.1.2 Formal Input Spiking Output layer

The FISO layer is designed to encode input data into spike trains according to the Send-on-Delta method (Section 7.1.1), except it does not feature lateral connections to inhibit all neurons when a spike is emitted. In contrast with the GenCell used in the previous chapter, the FISO layer combines both the spike encoding process and the first layer of neurons. This implementation relieves the architecture from the overhead imputed to the dedicated spike generation module. It should be noted that the concept of FISO neuron was first proposed during the early stages of the thesis. At that time, we were not aware of the S2NET-like SNN models, and the FISO neuron was intended to facilitate hybridization between Formal Neural Networks and rate-coded SNNs. This led to a patent filling in September 2019 concerning the concept of hybrid neural processing unit.

The FISO layer is implemented in parallel: there is one FISO neuron for each input neuron in the model. The architecture of the FISO neuron is depicted in Figure 7.4a. It derives closely from

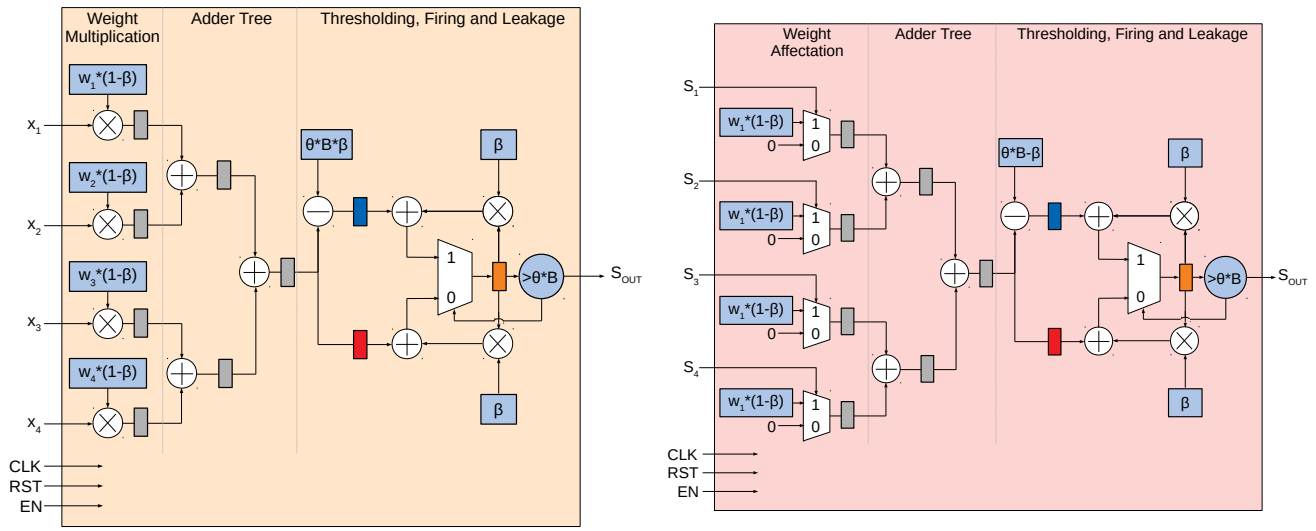


Figure 7.4: Hardware architecture of: a) FISO neuron with 4 input pixels, b) LIF neuron with 4 input synapses. Register barriers are shown in gray or colored rectangles. Data stored in memory is depicted in light blue squares.

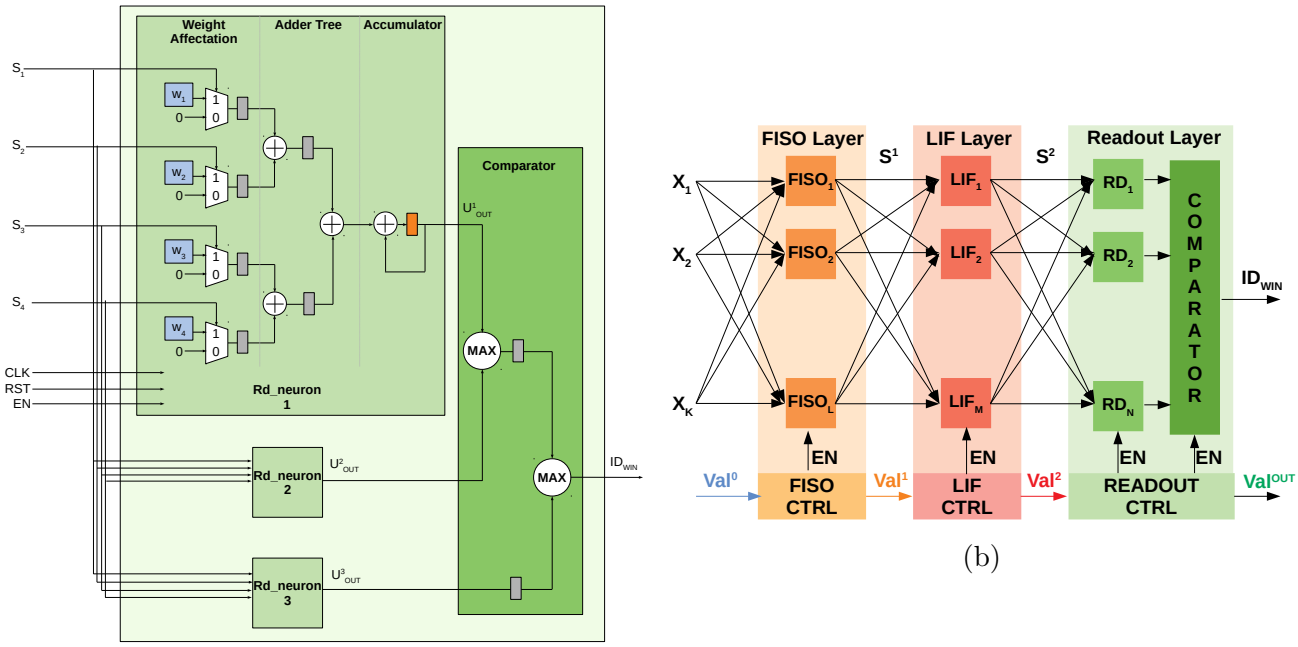
the Generic Neural Processing Unit (NPU) of PADS-V1. We only address the differences between the FISO neuron and the PADS-V1 NPU in this section. For more information on the hardware implementation, refer to the description of PADS-V1 NPU in Section 3.1.2. The main differences are the following: first, the weight affectation stage, which is replaced by a weight multiplication stage. Indeed, the integration of analog input requires to multiply weight with input activations. Second, the membrane potential decay: the membrane potential (orange register in Figure 7.4a) is multiplied with the decay factor (β) before weight accumulation (blue and red registers). The other differences have no influence on the implementation: the multiplication of synaptic weights with $1 - \beta$ is performed off-line, and so are the products $\theta \times B \times \beta$ and $\theta \times B$. Moreover, the architecture features an *enable* port to ensure synchronization in the pipeline. This mechanism is described in a dedicated paragraph at the end of this section.

7.3.1.3 LIF layer

The LIF neuron is similar to the FISO neuron, except that it features a weight affectation stage like in the original PADS-V1, since the inputs are binary signals. The architecture of the LIF neuron is given in Figure 7.4b. More detail on this implementation can be found in the description of PADS-V1 NPU in Section 3.1.2.

7.3.1.4 Readout layer

The Readout layer includes both the last layer of neurons and the selection of the winning class. It is illustrated in Figure 7.5a with 4 input synapses and 3 output neurons. The neurons are non-leaky and non-firing neurons which forwards their membrane potential to a comparator. The comparator is a pipelined tree of maximum operators designed to retrieve the maximum membrane potential (*i.e.* most active neuron). The *CLK* (clock), *RST* (reset) and *EN* (enable) signals are only depicted in the first readout neuron for the sake of simplicity.



(a) (b)

Figure 7.5: a) Architecture of the Readout layer with 4 input synapses and 3 neurons. b) Overview of the full architecture of PADS V2.

7.3.1.5 Pipeline & synchronization

A global overview of an MLP with three layers is depicted in Figure 7.5b. In order to ensure the synchronization of the pipeline, each layer is associated with a control unit. To better illustrate this matter, a chronogram of the network process is provided in Figure 7.6. The layers are synchronized using a *valid* signal. On the chronogram, the pipeline depth of the FISO layer is two clock cycles. Therefore, the FISO neurons starts processing input upon receiving the input *valid* signal (val^0). When data reaches the end of the pipeline (after two clock cycles), the output *valid* signal (val^1) is raised. Moreover, the valid signal remains up for the duration of the time-window, *e.g.* three clock cycles for three time-steps in the chronogram. This mechanism is repeated in each successive layer to ensure an end-to-end pipeline from spike encoding (FISO layer) to decoding (Readout layer). It should be noted that the pipeline depth in Figure 7.6 have been arbitrarily fixed to two, three and four clock cycles. The real pipe-line depth can be calculated depending on the layer configuration, as described in the next section.

7.3.2 Inference Time Results

The PADS-V2 architecture proposed in the previous section enables drastic acceleration compared to PADS-V1. This is achieved by both the timestep constraint and the fully-parallel spike encoding. In PADS-V1, the Spike Generation Cell is a dedicated module which encodes information sequentially. The sequential choice was made to limit the resource overhead caused by spike encoding: preliminary experiments have shown that a fully-parallel Spike Generation Cell for OPS-SAT dataset already saturates the Zedboard resources. In PADS-V2 the spike encoding is parallel, but it is directly supported by the first layer of neurons (FISO layer). In doing so, the cost of parallelization is mitigated. Evaluation of the resource cost of PADS-V2 should be

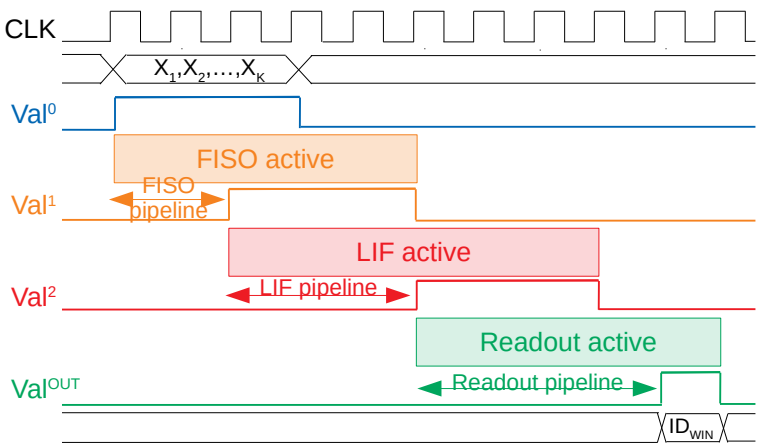


Figure 7.6: Chronogram of the PADS-V2 pipelined process at network-level

undertaken in further work. However, preliminary experiments shows that this implementation is much faster than PADS-V1. Both architectures are applied to two different tasks: a three-layer MLP (784-100-10) for MNIST classification, and a smaller MLP (80-10-2) for hybrid OPS-SAT classification (Chapter 6). The inference times are listed in Table 7.3.

Moreover, we provide the width of spike encoding window for both PADS-V1 and PADS-V2, for all datasets of the benchmark. Those values correspond to the best case accuracy of each method, and it is averaged on 10 samples. The results are shown in Table 7.4. In those results, the width of the spike encoding window is often smaller for rate-encoding in terms of timesteps. However, the sequential GenCell implies a much higher width in terms of clock cycle, since a timestep lasts for several clock-cycles (Chapter 3.1.1). As a reminder, parallelization of the Spike Generation Cell was found to be very resource-intensive in Chapter 3.1.1. Therefore, reducing the number of clock cycles for spike encoding through parallelization of the GenCell is not a viable solution. In PADS-V2 on the other hand, the width in timesteps is equal to the width in clock cycle, thanks to the fully-parallel spike encoding. In this case, parallelization is accessible since i) the spike generation is incorporated among a layer of LIF neurons ii) the SoD generation process is much simpler than rate coding in terms of hardware. Therefore, PADS-V2 spike encoding is temporally compact, reducing the overall inference-time by several order of magnitudes compared to PADS-V1. Moreover, the spike-encoding window width is a user-defined parameter in PADS-V2, whereas it is unpredictable and data-dependent in PADS-V1. Thus, the number of timesteps can grow out of control, like for RadioML-2018 in Table 7.4. In PADS-V2, the width of the window is fixed in advance, thus the problem does not arise. The variability of inference time in PADS-V1 is also a significant drawback because it complicates its integration in wider systems: in PADS-V2, the predictable inference time is much easier for timing management.

The preliminary experiments show that PADS-V2 is indeed much faster than PADS V1. That is thanks to the timestep-constrained representation of the SNN model, and to the end-to-end pipeline and fully-parallel implementation. Moreover, PADS-V2 architecture is much less influenced by input size than PADS-V1, as the spike encoding process is no longer dependent on input size. Thus, the new model of SNN seems much more adapted to hardware acceleration than rate-coded converted models studied in the previous chapters. Indeed, inference time was found to be the major source of energy overhead in spiking accelerators in Chapters 5 and 6. The timestep-constrained SNN model presented in this chapter, and the PADS-V2 architecture, could

Table 7.3: Inference time of PADS-V1 and PADS-V2 on MNIST and OPS-SAT hybrid

	PADS V1	**PADS V2**
MNIST 784-100-10	∼15 000 clk	39 clk
OPS-SAT (Hybrid) 80-10-2	∼120 clk	27 clk

Table 7.4: Width of the spike encoding window in PADS-V1 and PADS-V2 on the benchmark of datasets, for best-case accuracy. The width is expressed in both timesteps and clock-cycles.

	Spike encoding window width			
	PADS-V1 best acc.		**PADS-V2 best acc.**	
	TS	**CLK**	**TS**	**CLK**
MNIST	5	3920	10	10
GTSRB	5	15360	10	10
CIFAR-10	14	43008	10	10
MVR	9	540	5	5
Spoken Digits	10	5070	20	20
RadioML 2018	80	245760	20	20

therefore leverage significant energy savings. However, further work is required to deploy and test the architecture in hardware, in order to evaluate the resource, power and energy consumption of this architecture. Moreover, PADS-V2 is a prototype architecture and an operational design should involve a lower level of parallelism.

Additionally, the inference time in PADS-V2 is fully deterministic and only depends on the layer configuration. That is a major difference with PADS-V1, in which inference time is unpredictable and variable from one sample to another. This features facilitates the implementation of PADS-V2 in wider systems, easing timing management between the components. The inference time of PADS-V2 depends on the pipeline depth of each layer, as shown in the chronogram of Figure 7.6. The total execution time is expressed in Equation 7.8:

$$\Delta T^{\text{Total}} = \Delta T^{\text{FISO}}_{\text{pipe}} + \sum_{j}^{N^{\text{hidden}}} \Delta T^{\text{j-LIF}}_{\text{pipe}} + \Delta T^{\text{RD}}_{\text{pipe}} + N^{\text{ts}} \tag{7.8}$$

Where ΔT^{Total} is the total execution time (in clock cycles), $\Delta T^{\text{layer}}_{\text{pipe}}$ is the pipe-line depth (in clock cycles) of a given layer, N^{hidden} is the number of hidden layers of LIF neurons and N^{ts} is the number of time-steps for sample encoding. For FISO and LIF layers, the pipeline depth is computed as shown in Equation 7.9. It is the depth of the adder-tree, plus three clock cycles for threshold, firing and leakage.

$$\Delta T^{\text{FISO\_LIF}}_{\text{pipe}} = log2(N^{\text{input}}) + 3 \tag{7.9}$$

Where N^{input} is the number of input synapses. The pipeline depth of the readout layer is computed like shown in Equation 7.10. This time, the processing is divided in two separate pipe-line. For

readout neurons, the pipeline depth is the depth of the adder-tree, as there is no threshold, firing and leakage. After that, the pipeline depth of the comparator is the depth of the tree of maximum operators.

$$\Delta T_{\text{pipe}}^{\text{Readout}} = log2(N^{\text{input}}) + log2(N^{\text{output}}) \tag{7.10}$$

Where N^{output} is the number of output neurons. Those equations have been validated through hardware simulation.

7.4 Conclusions & Outlooks

In this chapter, we have described a novel model of SNNs proposed by Neftci *et. al.* in [32]. The model represents the network of LIF neurons as a Recurrent Neural Network (RNN) in order to use a variant of the Backpropagation Through Time algorithm: the Surrogate Gradient Learning. This model was adapted and simplified by Zimmer *et. al.* in the S2NET framework, which provides GPU-accelerated Surrogate Gradient Learning on timestep-constrained SNNs. This model uses the Send-on-Delta spike encoding method, whose parameters are also learned during training. Thanks to those features, the resulting SNN demonstrates lower Synaptic Activity Ratio than conversion and rate-coding on the benchmark of static datasets (Chapter 4). That is while maintaining acceptable or better accuracy than the previous technique used in this thesis. Moreover, the literature now describes state-of-the-art accuracy using this method.

The PADS architecture has been adapted to the S2NET SNN model, notably through adding a leakage factor to the membrane potential and integrating spike encoding in a layer of FISO neurons. PADS-V2 benefits from the fixed and low number of timesteps thanks to a fully-parallel implementation and an end-to-end pipeline. In doing so, the architecture achieves drastically reduced inference times compared to PADS-V1. On a 784-100-10 MLP applied to MNIST dataset, PADS-V2 demonstrates a constant inference time of 39 clock cycles per image. On the other hand, PADS-V1 implies variable inference time for an average 15 000 clock cycles. Thus, the time-constrained SNN model seems more adapted to hardware implementation, and could leverage drastic energy savings. However, further work is required to deploy the architecture in hardware and measure resource, power and energy consumption.

Moreover, PADS-V2 remains a prototype: an operational architecture should involve an intermediate level of parallelism. Indeed, a fully-parallel implementation like PADS is not realistic to deploy state-of-the-art models with several millions of neurons. Like explained in Chapter 5, parallelism can be tailored at layer level according to SAR: layers with higher SAR are implemented with more parallel NPUs than others. Such intermediate level of parallelism will be addressed in further works.

Chapter 8

Conclusions and outlooks

The field of Neuromorphic Computing has emerged recently as an answer to the energy requirements of Deep Learning models in view of their deployment in embedded systems. Deep neural networks are particularly appealing for autonomous and smart devices. Through the use of bio-inspired spiking neurons, the goal of Neuromorphic Computing is to leverage energy savings in hardware neural network accelerators. This widespread approach in the literature relies on the strong hypothesis that the the biological brain draws its energy efficiency from spike-based processing. Those expectations are backed by the respective computational costs of formal and spiking neurons: the accumulation operation of the latter is more hardware-friendly than the multiplication-accumulation operation of formal neurons. Moreover, the sparsity of spike encoding enables a sparse computation, *i.e.* the system is only active upon receiving information. That is, in contrast with FNN in which computation does not depend on data. Additionally, the lightweight communication between neurons should further reduce the hardware footprint of spiking implementations. However, to the best of our knowledge, there was a lack of fair, extensive and quantitative comparison between formal and spiking accelerators in the literature. This matter was the main topic of the thesis: determine and quantify the potential energy savings offered by Neuromorphic Computing.

In the literature, it appeared that the number of spikes generated by the network (*i.e* spiking activity) is often considered as a reliable high-level metric to assess energy consumption of spiking hardware accelerahttps://manga-encyclopedie.fandom.com/fr/wiki/Vegetotors relatively to an FNN baseline. This approach is based on the number of synaptic operations (MAC in formal neurons, ACC in spiking ones). However, the correlation between this metric and the actual energy consumption of SNNs remained unclear, and unproven when we started our study. Therefore, one other goal of this thesis was to propose a finer energy consumption model based on synaptic activity and confront the results with actual energy measurements.

Additionally, the last important goal of this thesis was to propose a cartography of applications and neural coding domains, in order to find specific use-cases in which neuromorphic acceleration could bring game-changing energy savings for deployment in embedded systems. In addition to exploring application cases, we were interested in finding "conditions" that enhance the energy savings of spiking implementations. Those conditions vary from the resolution of input data, the quantity of classes, the distribution of pixels... In other words, one of our concerns was to determine what makes an application suitable for spiking domain.

8.1 Conclusion

In this thesis, we have chosen to primarily address rate-coded Spiking Neural Networks and conversion training techniques, since those two methods were the most widespread and documented in the literature. Moreover, this family of SNNs is supported by several machine learning frameworks like N2D2. Several contributions were proposed to fulfill the thesis goals and answer its initial

questions.

First, we have proposed a high-level energy estimation model for FPGA neural network accelerators. This model is based on the Synaptic Activity Ratio metric (number of spike per synapse), and a modeling of MAC and ACC implementations on FPGA. The model proposes to compute the ratio of energy consumption between a MAC and ACC operations in a specific design, for a specific target board. This value, λ, depends on the saturation of DSP resources in the FPGA. If the average number of spike per synapses (SAR) is greater than λ, this means that the model is not suited for spiking acceleration. Thanks to this comparison method, one can rapidly determine which type of application seems suitable for spiking acceleration. The model was confronted to energy estimations on a few datasets. On most tested cases, for fully-parallel architectures, the SAR&λ model accurately predicted the preferable coding domain in terms of energy consumption. That was not the case for fully-sequential architectures, since the two implementations (SPLEAT and C-HLS) have two much different low-level design choices (pipeline). On the other hand, the model is not quantitatively correlated with the estimated ratios of energy consumption. Our hypothesis is that the model does not take the temporal sparsity of spikes into account, which is a strong bias considering that the tested architectures maintain their power consumption in idle state (*i.e.* between two spikes). This matter will be addressed further in the outlooks.

Second, we have proposed a framework for estimation of logic resource usage, power consumption, inference time and energy of FPGA neural network accelerators. This framework is based on the interpolation of a hardware-footprint database and low level simulation of spiking inference. It is able to determine the hardware footprint of a typical spiking or formal hardware accelerator, under high or low levels of parallelism. In doing so, the goal is not to provide an absolute estimation of hardware footprint, but rather to facilitate the rapid comparison of coding domains and levels of parallelism in the application space. Thanks to the framework, the cartography of neural coding domain and architectural choices on a benchmark of seven datasets was possible in a few hours. Without the framework, this study could have taken hundreds of hours of synthesis, simulation and result extraction. Using our framework, the extensive comparison of hardware FNNs and SNNs has shown that spikes are not always better in terms of energy consumption (far from it). Indeed, the temporal sparsity of rate-coded spike trains causes a substantial overhead in inference time, and in turns, in energy consumption. To cope with this issue, specific strategies have to be settled, like reducing temporal sparsity of spikes or number of spikes per synapses. The estimation framework led to a publication in ACM Transaction on Embedded Computing Systems, Special Issue : Accelerating AI on the edge [112].

To answer this matter, we have proposed two different approaches. On the one hand, this thesis is among the first to study the concept of neural network hybridization. The motivations for this approach are numerous: tailoring the neural coding domain to layer-wise synaptic activity, using the formal feature-extraction stage as a pre-processing to lower resolution and widen data distribution. Both energy estimations and hardware implementations demonstrate that this approach could indeed leverage energy savings where spiking domain alone could not. Our hybrid architecture led to a publication in ISCAS 2020 conference [113], and to a use-case chapter in the Tulipp Book [114]. On the other hand, we use our high-level SAR-based energy model to find a more suitable family of SNN models. In doing so, we address the timestep-constrained SNNs trained through Surrogate Gradient Learning proposed for the very first time in 2019 by Neftci *et. al.*. This family of model demonstrates lower SAR and spike temporal sparsity than rate-based approaches, while offering state-of-the-art classification accuracy. We proposed an hardware accelerator based on this family of SNN models (PADS-V2), and obtained substantially lower inference time than with the former rate-coding and conversion approach. Hence, this approach seems

promising to reduce the energy consumption of neuromorphic implementations. In PADS V2 implementation, we used the FISO neuron model which was patented during the first year of this thesis. The PADS V2 implementation will be the main topic of an upcoming conference paper.

Additionally, this thesis has led to the deployment of the first ever neuromorphic accelerator in space, as well as the first hybrid accelerator, on-board ESA's OPS-SAT experimental satellite. The satellite was launched in December 2019, and the VGT-PADS hybrid architecture (Chapter 6) was successfully tested in-flight. This world premiere was achieved in collaboration with IRT Saint-Exupéry and CIAR project. This led to a publication for the European OBDP workshop [93].

Lastly, one of the main assessments of this work is that rate-coding is not suited to energy efficiency in digital neuromorphic circuits compared to classical floating-point approaches. As demonstrated in Chapter 5, rate-coding often leads to an energy consumption overhead, whether due to the number of operations (*i.e.* number of spikes) or to the temporal sparsity of spikes. The latter implies a higher inference time, and an energy overhead due to both idle and static power consumptions (particularly on FPGAs). However, those conclusions are not inevitably true for SNNs in general, since other methods than rate-coding exists. Other encoding schemes should be studied in this regard, such as temporal [68] or rank-order [66] coding. While reducing the number or temporal sparsity of spikes, those methods are also more biologically plausible. They indeed reproduce the mechanism observed in retina sensory fibers since they are interpreted as intensity-to-latency encoders [115]. Addressing temporal and rank-order encoding is therefore coherent with the bio-inspiration approach of neuromorphic engineering, in our hunt for brain-inspired energy efficiency.

8.2 Outlooks

All along this document, retrospective analysis of our development and results have led to several short-term outlooks. Those propositions are either intended to strengthen our results, or to delve deeper on some aspects that were left apart.

8.2.1 Short term and work-specific perspectives

Since PADS and SPLEAT are two prototypes with extreme (high and low) levels of parallelism, those are only laboratory prototypes unsuitable for state-of-the-art model emulation or operational deployment. Hence, one of the first short-term development to make is the development of a neuromorphic accelerator featuring an intermediate, configurable levels of parallelism. For example, such an architecture might be based on the SPLEAT Neural Processing Core, which is already adapted to multiplexing. The goal is therefore to deploy several SPLEAT cores in parallel for each layer. Since we have found that SAR was an indicator of parallelism requirements, it can be used to determine the optimal number of parallel cores in each layers. Thus, further studies on the relation between SAR and parallelism requirements will be addressed in further works. Similarly, the exploration of intermediate levels of parallelism might also be a future feature of the hardware-footprint and energy estimation framework. In doing so, the framework can estimate energy consumption for parallelism configuration, allowing to find a Pareto-optimal solution. Additionally, the architecture should be adapted to the timestep-constrained SNN model [32] [21] studied in Chapter 7, rather than the original rate-based model it was designed for. The goal is indeed to benefit from the temporal compactness and low SAR of such novel SNN models.

Moreover, we are also interested in improving further the SAR metric and the subsequent high-level energy model. The SAR metric must take the temporal sparsity of spikes into account to provide a quantitative estimation of energy ratios. Indeed, the current approach neglects the idle power usage of hardware architectures, which represents an important part of the overall energy consumption. Indeed, our measurements have shown that the idle dynamic power usage (*i.e.* dynamic component of power consumption of a neural core waiting for input spikes) was nearly as high as the active dynamic power usage (*i.e.* dynamic component of a neural core actively processing spikes). That is mostly due to the clock toggle signals, which are independent from the presence or absence of input stimuli. In temporally sparse SNNs, idle state represents a large portion of time in each neurons, which should be accounted for in the SAR metric. The effect of idle state is less of a problem in temporally compact SNNs. However, if there is no mean to decrease temporal sparsity of spikes in a given application, the problem might also be addressed the other way around. Instead of adapting the metric, a clock-gating mechanism in the architecture could preclude the influence of dynamic power consumption in idle state. By deactivating the clock signals locally when a core is in idle state, such low-level optimization could provide substantial energy savings in neuromorphic implementations. Moreover, the result would be closer to the SAR-based model.

Additionally, this study has shown that FPGA was not the best suited device for neuromorphic acceleration. That is because of the large static power consumption which has two noticeable effects. First, the static power offset masks the dynamic power savings, particularly for small designs in which the dynamic component of power consumption is negligible in front of the other (on Zedboard). Preliminary experiments targeting other and larger boards have shown the same issue, if not worse. Second, a high static power consumption further penalizes the energy cost of idle state in temporally sparse SNNs (rate-coding & conversion). Clock-gating can drastically reduce the dynamic power usage in idle state, but not the static component. One way to cope with this issue is to address Application Specific Integrated Circuits (ASICs), which demonstrate much lower static power consumption. ASICs are integrated circuits specifically manufactured according to the RTL description, in contrast with FPGAs which are array of re-configurable elements. The hardware of ASICs being dedicated and specific, the static power consumption can be much lower. The deployment of neuromorphic accelerators on ASICs rather than FPGAs could therefore enhance the low-power capabilities of spiking neurons, and in turns leverage the expected energy savings.

Furthermore, this study focuses on feed-forward CNN architectures. In future works, we will extend our cartography of neural coding domains to other models, such as Recurrent Neural Network, Reservoir Computing or Residual architectures. Lastly, one of the most promising application of Spiking Neural Networks are their use in combination with Dynamic Vision Sensors (*i.e.* Event-based cameras). Therefore, the neuromorphic acceleration applied to event-based sensors will be addressed in further work.

8.2.2 Middle term outlooks and insights

We certainly still have a lot to learn from the energy efficiency of our biological brain. Since it is the most efficient (by far) neural processor anyone has ever heard of, taking inspiration from Neurosciences models must be a fruitful approach. The most important thing to learn from this thesis is: using a naive approach of hardware implementation is not sufficient to benefit from the energy efficiency of spike encoding. Three major necessary conditions to take advantage from spikes were identified in this work.

- First, the number of spikes can compensate the energy efficiency of the ACC operation over the MAC. Therefore, the SAR should be studied to evaluate the potential energy savings brought by spiking acceleration for a given neural network model and application.
- Second, one must ensure of the temporal compactness of spike encoding, since the idle energy consumption (*i.e.* waiting between spikes) can compensate the energy savings obtained on active processing.
- Third, when temporal sparsity of spikes is not an option (in specific application cases for example), systems should involve a clock-gating mechanism to preclude idle energy consumption.
- Fourth and last, ASIC technology seems more suited to neuromorphic acceleration than FPGAs. That is thanks to the lower static power consumption of ASICs. Indeed, this feature enhances the dynamic power savings offered by the spiking synaptic operation, and further reduces the idle dynamic power consumption.

When looking towards ASICs and power-gating techniques, a particularly appealing emerging technology arises from literature: Non Volatile RAM devices (NVRAM) [116]. This technology is able to retain information when turned off like an hard drive, but offers much faster read and right accesses. Using this RAM technology, parts of the system can be instantly turned off and on for a strong and efficient power-gating: idle neurons can be turned-off while waiting for input spikes. Moreover, NVRAMs demonstrate very low, if not negligible energy consumption compared to standard RAM technologies. Some specific types of NVRAMs, such as RRAMs [117] or STT-RAMs [118] [119], have even more appealing features for Neuromorphic Computing. Indeed, such devices are able to physically emulate synapses: their internal resistivity can be tuned according to the intensity of current passing through. This mechanism is analogous to synaptic plasticity, and this devices are better known under the name of Memristor [120]. Using Memristor crossbar arrays, researchers are currently working on emulating simple neural networks and even local learning rules like Spike Time Dependent Plasticity [121] [122] [123]. This technology promises a breakthrough reduction of energy consumption in neuromorphic systems [124] [125]. However, this field of research is still in its very early stages, since the Memristor technology itself is very immature. The intrinsic variability of the production method, and the difficulty to stabilize them on large scales are still limiting the technology deployment. The latest experiments on Memristor crossbars only involve a few dozens of synapses, which is far from supporting basic CNN models, let alone state-of-the-art Deep Learning topologies. Still, the Logic-in-Memory approach of this emerging field of research could be the key to the brain-inspired energy efficiency.

Bibliography

[1] O. e. a. Bichler, "N2d2-neural network design & deployment," Manual available on Github, 2017.

[2] Q. Zhang, M. Zhang, T. Chen, Z. Sun, Y. Ma, and B. Yu, "Recent advances in convolutional neural network acceleration," Neurocomputing, vol. 323, pp. 37–51, 2019.

[3] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: Comparison of fpga, cpu, gpu, and asic," in 2016 International Conference on Field-Programmable Technology (FPT). IEEE, 2016, pp. 77–84.

[4] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. Ong Gee Hock, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra et al., "Can fpgas beat gpus in accelerating next-generation deep neural networks?" in Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2017, pp. 5–14.

[5] M. K. Hamdan, "Vhdl auto-generation tool for optimized hardware acceleration of convolutional neural networks on fpga (vgt)," Ph.D. dissertation, Iowa State University, 2018.

[6] E. M. Izhikevich, "Which model to use for cortical spiking neurons?" IEEE transactions on neural networks, vol. 15, no. 5, pp. 1063–1070, 2004.

[7] L. A. Camuñas-Mesa, Y. L. Domínguez-Cordero, A. Linares-Barranco, T. Serrano-Gotarredona, and B. Linares-Barranco, "A configurable event-driven convolutional node with rate saturation mechanism for modular convnet systems implementation," Frontiers in neuroscience, vol. 12, p. 63, 2018.

[8] R. M. Wang, G. Cohen, K. M. Stiefel, T. J. Hamilton, J. C. Tapson, and A. van Schaik, "An fpga implementation of a polychronous spiking neural network with delay adaptation," Frontiers in neuroscience, vol. 7, p. 14, 2013.

[9] X. Ju, B. Fang, R. Yan, X. Xu, and H. Tang, "An fpga implementation of deep spiking neural networks for low-power and fast classification," Neural computation, vol. 32, no. 1, pp. 182–204, 2020.

[10] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated systolic array architecture synthesis for high throughput cnn inference on fpgas," in Proceedings of the 54th Annual Design Automation Conference 2017, 2017, pp. 1–6.

[11] H. Fang, Z. Mei, A. Shrestha, Z. Zhao, Y. Li, and Q. Qiu, "Encoding, model, and architecture: Systematic optimization for spiking neural network in fpgas," in 2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD). IEEE, 2020, pp. 1–9.

[12] G. Orchard, C. Meyer, R. Etienne-Cummings, C. Posch, N. Thakor, and R. Benosman, "Hfirst: A temporal approach to object recognition," IEEE transactions on pattern analysis and machine intelligence, vol. 37, no. 10, pp. 2028–2040, 2015.

[13] A. Khodamoradi, K. Denolf, and R. Kastner, "S2n2: A fpga accelerator for streaming spiking neural networks," in The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2021, pp. 194–205.

[14] N. Abderrahmane, "Hardware design of spiking neural networks for energy efficient brain-inspired computing," Ph.D. dissertation, Université Côte d'Azur, 2020.

[15] P. S. Lacoste, "Deux premières en ia embarquée à bord de satellites • irt saint exupéry • technological research institute." [Online]. Available: https://www.irt-saintexupery.com/fr/two-premieres-in-on-board-artificial-intelligence-on-satellites/

[16] R. P. e. a. Gorman, "Analysis of hidden units in a layered network trained to classify sonar targets," Neural networks, vol. 1, no. 1, pp. 75–89, 1988.

[17] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," arXiv preprint arXiv:1804.03209, 2018.

[18] T. J. O'Shea, T. Roy, and T. C. Clancy, "Over-the-air deep learning based radio signal classification," IEEE Journal of Selected Topics in Signal Processing, vol. 12, no. 1, pp. 168–179, 2018.

[19] [Online]. Available: https://directory.eoportal.org/web/eoportal/satellite-missions/o/ops-sat#?TMYJ1faHerb

[20] "Ops-sat." [Online]. Available: https://www.esa.int/Enabling_Support/Operations/OPS-SAT

[21] R. Zimmer, T. Pellegrini, S. F. Singh, and T. Masquelier, "Technical report: supervised training of convolutional spiking neural networks with pytorch," arXiv preprint arXiv:1911.10124, 2019.

[22] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," The bulletin of mathematical biophysics, vol. 5, no. 4, pp. 115–133, 1943.

[23] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." Psychological review, vol. 65, no. 6, p. 386, 1958.

[24] P. Werbos, "Beyond regression:" new tools for prediction and analysis in the behavioral sciences," Ph. D. dissertation, Harvard University, 1974.

[25] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," nature, vol. 323, no. 6088, pp. 533–536, 1986.

[26] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," Neural computation, vol. 1, no. 4, pp. 541–551, 1989.

[27] Y. LeCun, Y. Bengio et al., "Convolutional networks for images, speech, and time series," The handbook of brain theory and neural networks, vol. 3361, no. 10, p. 1995, 1995.

[28] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, 2014.

[29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in Advances in neural information processing systems, 2017, pp. 5998–6008.

[30] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell et al., "Language models are few-shot learners," arXiv preprint arXiv:2005.14165, 2020.

[31] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going deeper in spiking neural networks: Vgg and residual architectures," Frontiers in neuroscience, vol. 13, p. 95, 2019.

[32] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks," IEEE Signal Processing Magazine, vol. 36, no. 6, pp. 51–63, 2019.

[33] M. Miskowicz, "Send-on-delta concept: An event-based data reporting strategy," sensors, vol. 6, no. 1, pp. 49–63, 2006.

[34] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," Computer Science Review, vol. 3, no. 3, pp. 127–149, 2009.

[35] W. Maass, "Liquid state machines: motivation, theory, and applications," Computability in context: computation and logic in the real world, pp. 275–296, 2011.

[36] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," Neural networks, vol. 2, no. 5, pp. 359–366, 1989.

[37] Y. LeCun et al., "Lenet-5, convolutional neural networks," URL: http://yann. lecun. com/exdb/lenet, vol. 20, no. 5, p. 14, 2015.

[38] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in International conference on machine learning. PMLR, 2015, pp. 448–456.

[39] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in 2013 IEEE international conference on acoustics, speech and signal processing. Ieee, 2013, pp. 6645–6649.

[40] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in Thirty-first AAAI conference on artificial intelligence, 2017.

[41] B. Asadi and H. Jiang, "On approximation capabilities of relu activation and softmax output layer in neural networks," arXiv preprint arXiv:2002.04060, 2020.

[42] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[43] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," nature, vol. 521, no. 7553, pp. 436–444, 2015.

[44] A. Joubert, B. Belhadj, O. Temam, and R. Héliot, "Hardware spiking neurons design: Analog or digital?" in The 2012 International Joint Conference on Neural Networks (IJCNN). IEEE, 2012, pp. 1–5.

[45] J. Von Neumann, "The general and logical theory of automata." 1951.

[46] T. S. Crow, "Evolution of the graphical processing unit," A professional paper submitted in partial fulfillment of the requirements for the degree of Master of Science with a major in Computer Science, University of Nevada, Reno, 2004.

[47] M. J. S. Smith, Application-specific integrated circuits. Addison-Wesley Reading, MA, 1997, vol. 7.

[48] S. M. Trimberger, Field-programmable gate array technology. Springer Science & Business Media, 2012.

[49] R. W. Hartenstein, Hardware description languages. North-Holland, 1987.

[50] Z. Navabi, VHDL: Analysis and modeling of digital systems. McGraw-Hill New York, 1993, vol. 2.

[51] D. Thomas and P. Moorby, The Verilog® hardware description language. Springer Science & Business Media, 2008.

[52] A. Thomas, "Memristor-based neural networks," Journal of Physics D: Applied Physics, vol. 46, no. 9, p. 093001, 2013.

[53] S. Mittal, "A survey of fpga-based accelerators for convolutional neural networks," Neural computing and applications, vol. 32, no. 4, pp. 1109–1139, 2020.

[54] L. F. Abbott, "Lapicque's introduction of the integrate-and-fire model neuron (1907)," Brain research bulletin, vol. 50, no. 5-6, pp. 303–304, 1999.

[55] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," The Journal of physiology, vol. 117, no. 4, pp. 500–544, 1952.

[56] R. Brette, "What is the most realistic single-compartment model of spike initiation?" PLoS Comput Biol, vol. 11, no. 4, p. e1004114, 2015.

[57] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," Neural Networks, vol. 111, pp. 47 – 63, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608018303332

[58] 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.

[59] P. O'Connor and M. Welling, "Deep spiking networks," arXiv preprint arXiv:1602.08323, 2016.

[60] K. S. Burbank, "Mirrored stdp implements autoencoder learning in a network of spiking neurons," PLoS computational biology, vol. 11, no. 12, p. e1004566, 2015.

[61] J. A. Leñero-Bardallo, T. Serrano-Gotarredona, and B. Linares-Barranco, "A 3.6μs latency asynchronous frame-free event-driven dynamic-vision-sensor," IEEE Journal of Solid-State Circuits, vol. 46, no. 6, pp. 1443–1455, 2011.

[62] V. Chan, S.-C. Liu, and A. van Schaik, "Aer ear: A matched silicon cochlea pair with address event representation interface," IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 54, no. 1, pp. 48–59, 2007.

[63] A. Slepyan and N. Thakor, "Towards scalable soft e-skin: Flexible event-based tactile-sensors using wireless sensor elements embedded in soft elastomer," in 2020 8th IEEE RAS/EMBS International Conference for Biomedical Robotics and Biomechatronics (BioRob). IEEE, 2020, pp. 334–339.

[64] 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.

[65] T. Gollisch and M. Meister, "Rapid neural coding in the retina with relative spike latencies," science, vol. 319, no. 5866, pp. 1108–1111, 2008.

[66] G. Portelli, J. M. Barrett, G. Hilgen, T. Masquelier, A. Maccione, S. Di Marco, L. Berdondini, P. Kornprobst, and E. Sernagor, "Rank order coding: a retinal information decoding strategy revealed by large-scale multielectrode array retinal recordings," Eneuro, vol. 3, no. 3, 2016.

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

[68] B. Rueckauer and S.-C. Liu, "Conversion of analog to spiking neural networks using sparse temporal coding," in 2018 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 2018, pp. 1–5.

[69] W. Fang, Z. Yu, Y. Chen, T. Masquelier, T. Huang, and Y. Tian, "Incorporating learnable membrane time constant to enhance learning of spiking neural networks," in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 2661–2671.

[70] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," Proceedings of the IEEE, vol. 105, no. 12, pp. 2295–2329, 2017.

[71] J. C. Thiele, O. Bichler, and A. Dupret, "Event-based, timescale invariant unsupervised online deep learning with stdp," Frontiers in Computational Neuroscience, vol. 12, p. 46, 2018. [Online]. Available: https://www.frontiersin.org/article/10.3389/fncom.2018.00046

[72] 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. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0893608017302903

[73] G. Srinivasan, C. Lee, A. Sengupta, P. Panda, S. S. Sarwar, and K. Roy, "Training deep spiking neural networks for energy-efficient neuromorphic computing," in ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2020, pp. 8549–8553.

[74] S. M. Bohte, J. N. Kok, and J. A. La Poutré, "Spikeprop: backpropagation for networks of spiking neurons." in ESANN, 2000, pp. 419–424.

[75] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," Neural Networks, vol. 111, pp. 47–63, 2019.

[76] P. Panda, S. A. Aketi, and K. Roy, "Toward scalable, efficient, and accurate deep spiking neural networks with backward residual connections, stochastic softmax, and hybridization," Frontiers in Neuroscience, vol. 14, 2020.

[77] S. Davidson and S. B. Furber, "Comparison of artificial and spiking neural networks on digital hardware," Frontiers in Neuroscience, vol. 15, p. 345, 2021.

[78] S. Kundu, G. Datta, M. Pedram, and P. A. Beerel, "Spike-thrift: Towards energy-efficient deep spiking neural networks by limiting spiking activity via attention-guided compression," in Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2021, pp. 3953–3962.

[79] M. Bouvier, A. Valentian, T. Mesquida, F. Rummens, M. Reyboz, E. Vianello, and E. Beigne, "Spiking neural networks hardware implementations and challenges: A survey," ACM Journal on Emerging Technologies in Computing Systems (JETC), vol. 15, no. 2, pp. 1–35, 2019.

[80] K. Cheung, S. R. Schultz, and W. Luk, "A large-scale spiking neural network accelerator for fpga systems," in International Conference on Artificial Neural Networks. Springer, 2012, pp. 113–120.

[81] E. M. Izhikevich, "Polychronization: computation with spikes," Neural computation, vol. 18, no. 2, pp. 245–282, 2006.

[82] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in 2015 International joint conference on neural networks (IJCNN). ieee, 2015, pp. 1–8.

[83] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain et al., "Loihi: A neuromorphic manycore processor with on-chip learning," Ieee Micro, vol. 38, no. 1, pp. 82–99, 2018.

[84] E. Painkras, L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, and S. B. Furber, "Spinnaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation," IEEE Journal of Solid-State Circuits, vol. 48, no. 8, pp. 1943–1953, 2013.

[85] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura et al., "A million spiking-neuron integrated circuit with a scalable communication network and interface," Science, vol. 345, no. 6197, pp. 668–673, 2014.

[86] D. Neil and S.-C. Liu, "Minitaur, an event-driven fpga-based spiking network accelerator," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 22, no. 12, pp. 2621–2628, 2014.

[87] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O'brien, Y. Umuroglu, M. Leeser, and K. Vissers, "Finn-r: An end-to-end deep-learning framework for fast exploration of quantized neural networks," ACM Transactions on Reconfigurable Technology and Systems (TRETS), vol. 11, no. 3, pp. 1–23, 2018.

[88] L. Khacef, N. Abderrahmane, and B. Miramond, "Confronting machine-learning with neuroscience for neuromorphic architectures design," in 2018 International Joint Conference on Neural Networks (IJCNN). IEEE, 2018, pp. 1–8.

[89] B. Han, A. Sengupta, and K. Roy, "On the energy benefits of spiking deep neural networks: A case study," in 2016 International Joint Conference on Neural Networks (IJCNN). IEEE, 2016, pp. 971–976.

[90] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," Advances in neural information processing systems, vol. 25, pp. 1097–1105, 2012.

[91] S. R. Kheradpisheh and T. Masquelier, "Temporal backpropagation for spiking neural networks with one spike per neuron," International Journal of Neural Systems, vol. 30, no. 06, p. 2050027, 2020.

[92] Xilinx, "Zynq dpu v3.2 - product guide," 07 2020.

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

[94] I. Goodfellow, Y. Bengio, and A. Courville, Deep learning. MIT press, 2016.

[95] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.

[96] A. Byerly, T. Kalganova, and I. Dear, "A branching and merging convolutional network with homogeneous filter capsules," arXiv preprint arXiv:2001.09136, 2020.

[97] J. e. a. Stallkamp, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," Neural networks, vol. 32, pp. 323–332, 2012.

[98] Á. e. a. Arcos-García, "Deep neural network for traffic sign recognition systems: An analysis of spatial transformers and stochastic optimisation methods," Neural Networks, vol. 99, pp. 158–165, 2018.

[99] A. Krizhevsky, G. Hinton et al., "Learning multiple layers of features from tiny images," 2009.

[100] P. e. a. Foret, "Sharpness-aware minimization for efficiently improving generalization. arxiv 2020," arXiv preprint arXiv:2010.01412.

[101] L. K. et. al., "Written and spoken digits database for multimodal learning," Oct. 2019.

[102] L. Zhejun, "Resnet for radio recognition," Jun 2019. [Online]. Available: https://github.com/liuzhejun/ResNet-for-Radio-Recognition

[103] Xilinx, "Zynq-7000 soc data sheet - overview," 7 2018.

[104] I. S. U. Muhammad K A Hamdan, "Automatic vhdl generation for cnn models," 05 2020.

[105] R. Bonamy, S. Bilavarn, D. Chillet, and O. Sentieys, "Power modeling and exploration of dynamic and partially reconfigurable systems," Journal of Low Power Electronics, vol. 12, no. 3, pp. 172–185, 2016.

[106] B. Rueckauer, I.-A. Lungu, Y. Hu, and M. Pfeiffer, "Theory and tools for the conversion of analog to spiking convolutional neural networks," arXiv preprint arXiv:1612.04052, 2016.

[107] Y. Li, S. Deng, X. Dong, R. Gong, and S. Gu, "A free lunch from ann: Towards efficient, accurate spiking neural networks calibration," arXiv preprint arXiv:2106.06984, 2021.

[108] M. A. Akhloufi, S. Arola, and A. Bonnet, "Drones chasing drones: Reinforcement learning and deep search area proposal," Drones, vol. 3, no. 3, p. 58, 2019.

[109] D. Evans and M. Merri, "Ops-sat: A esa nanosatellite for accelerating innovation in satellite control," in SpaceOps 2014 Conference, 2014, p. 1702.

[110] F. Davoli, C. Kourogiorgas, M. Marchese, A. Panagopoulos, and F. Patrone, "Small satellites and cubesats: Survey of structures, architectures, and protocols," International Journal of Satellite Communications and Networking, vol. 37, no. 4, pp. 343–359, 2019.

[111] Y.-L. Boureau, F. Bach, Y. LeCun, and J. Ponce, "Learning mid-level features for recognition," in 2010 IEEE computer society conference on computer vision and pattern recognition. IEEE, 2010, pp. 2559–2566.

[112] E. Lemaire, B. Miramond, S. Bilavarn, H. Saoud, and N. Abderrahmane, "Synaptic activity and hardware footprint of spiking neural networks in digital neuromorphic systems," ACM Transactions on Embedded Computing Systems (TECS), vol. 37, no. 4, p. 26, 2022.

[113] E. Lemaire, M. Moretti, L. Daniel, B. Miramond, P. Millet, F. Feresin, and S. Bilavarn, "An fpga-based hybrid neural network accelerator for embedded satellite image classification," in 2020 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 2020, pp. 1–5.

[114] M. Jahre, D. Göhringer, and P. Millet, Towards Ubiquitous Low-power Image Processing Platforms. Springer, 2021.

[115] E. D. Adrian and R. Matthews, "The action of light on the eye: Part i. the discharge of impulses in the optic nerve and its relation to the electric changes in the retina," The Journal of Physiology, vol. 63, no. 4, p. 378, 1927.

[116] A. Chen, "A review of emerging non-volatile memory (nvm) technologies and applications," Solid-State Electronics, vol. 125, pp. 25–38, 2016.

[117] W. Zhuang, W. Pan, B. Ulrich, J. Lee, L. Stecker, A. Burmaster, D. Evans, S. Hsu, M. Tajiri, A. Shimaoka et al., "Novel colossal magnetoresistive thin film nonvolatile resistance random access memory (rram)," in Digest. International Electron Devices Meeting,. IEEE, 2002, pp. 193–196.

[118] E. Kültürsay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating stt-ram as an energy-efficient main memory alternative," in 2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). IEEE, 2013, pp. 256–267.

[119] K. Wang, J. Alzate, and P. K. Amiri, "Low-power non-volatile spintronic memory: Stt-ram and beyond," Journal of Physics D: Applied Physics, vol. 46, no. 7, p. 074003, 2013.

[120] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," nature, vol. 453, no. 7191, pp. 80–83, 2008.

[121] M. Hu, H. Li, Y. Chen, Q. Wu, G. S. Rose, and R. W. Linderman, "Memristor crossbar-based neuromorphic computing system: A case study," IEEE transactions on neural networks and learning systems, vol. 25, no. 10, pp. 1864–1878, 2014.

[122] C. Li, Z. Wang, M. Rao, D. Belkin, W. Song, H. Jiang, P. Yan, Y. Li, P. Lin, M. Hu et al., "Long short-term memory networks in memristor crossbar arrays," Nature Machine Intelligence, vol. 1, no. 1, pp. 49–57, 2019.

[123] T. Serrano-Gotarredona, T. Masquelier, T. Prodromakis, G. Indiveri, and B. Linares-Barranco, "Stdp and stdp variations with memristors for spiking neuromorphic learning systems," Frontiers in neuroscience, vol. 7, p. 2, 2013.

[124] P. Wijesinghe, A. Ankit, A. Sengupta, and K. Roy, "An all-memristor deep spiking neural computing system: A step toward realizing the low-power stochastic brain," IEEE Transactions on Emerging Topics in Computational Intelligence, vol. 2, no. 5, pp. 345–358, 2018.

[125] P.-F. Chiu, M.-F. Chang, C.-W. Wu, C.-H. Chuang, S.-S. Sheu, Y.-S. Chen, and M.-J. Tsai, "Low store energy, low vddmin, 8t2r nonvolatile latch and sram with vertical-stacked resistive memory (memristor) devices for low power mobile applications," IEEE Journal of Solid-State Circuits, vol. 47, no. 6, pp. 1483–1496, 2012.

[126] M. Toumazet, "Ops-sat: Ai in the stars – an article by members • irt saint exupéry • technological research institute," Jan 2021. [Online]. Available: https://www.irt-saintexupery.com/ops-sat-ai-in-the-stars/

[127] Xilinx, "Petalinux tools documentation - reference guide," 07 2020.

APPENDIX 1: CIAR project

The CIAR (Autonomous and Reactive Image Chain) project [126] is a project led by Saint-Exupery Technical Research Institute (IRT) involving several private and public research partners, including ActiveEon, AViSTO, ELSYS Design, GEO4i, Thales Alenia Space and the LEAT laboratory. The CIAR project is dedicated to the development of Neural Network accelerators for on-board satellite image processing, targeting OPS-SAT Cyclone V SoC. This project has an access to the OPS-SAT platform provided by ESA. The CIAR project thus represents a valuable opportunity for architecture testing in real-world conditions.

The CIAR project use-case answers to a very concrete problem for observation satellites: the limited communication bandwidth between the satellite and its ground control. Indeed, imaging satellites role is to take high-resolution pictures of the Earth from space. Those pictures are then sent to the ground for exploitation. However, the bandwidth between the satellite and its ground station is very limited. Thus, sending high-resolution pictures is not a trivial concern and sending useless images should be avoided.

The specific use-case of the CIAR project is to automatically segment clouds on pictures. In doing so, the system is able to extract the parts where the ground is visible. Using this method, only useful information is sent to the ground to help relieve the communication bandwidth. Therefore, our goal is to build a cloud segmentation system and deploy it on the OPS-SAT SoC. The system separates full-size (1920x1080p) images in 28x28p patches, and classifies each as "*cloud*" or "*no-cloud*". The results are recombined after classification to form full-size cloud segmentation maps. The cloud segmentation process is illustrated in Figure 1. The original picture is on the left. On the right, patches classified as *cloud* appear in yellow.

Several Artificial Neural Networks models and hardware architectures have been developed to address the cloud classification task. Each model has been evaluated in terms of performance, resource and power usage, execution time and energy consumption. The goal is to determine which type of application would suit best to the cloud segmentation use-case in the context of highly constrained satellite systems. The various Neural Network models are:

- LeNet CNN architecture designed using VGT generator (Section 2.2.2.1),
- Spiking Multilayer Perceptron coded by hand in VHDL (PADS architecture),
- Tiny-Yolo CNN architecture generated using VGT,
- Fully-Convolutional Neural Network (FCN) architecture without fully-connected layers generated using VGT,
- Spiking CNN (S-CNN) coded by hand in VHDL.
- Hybrid Neural Network (HNN) mixing VGT-generated and hand-coded VHDL.

OPS-SAT LIST OF MODULES:

- The Satellite Experimental Processing Platform (SEPP): an Altera Cyclone V System-on-Chip (SoC), alsa called MitySOM, with large memory capabilities in order to support advanced software and hardware experiments. This SoC is made of an FPGA Fabric, a dual-

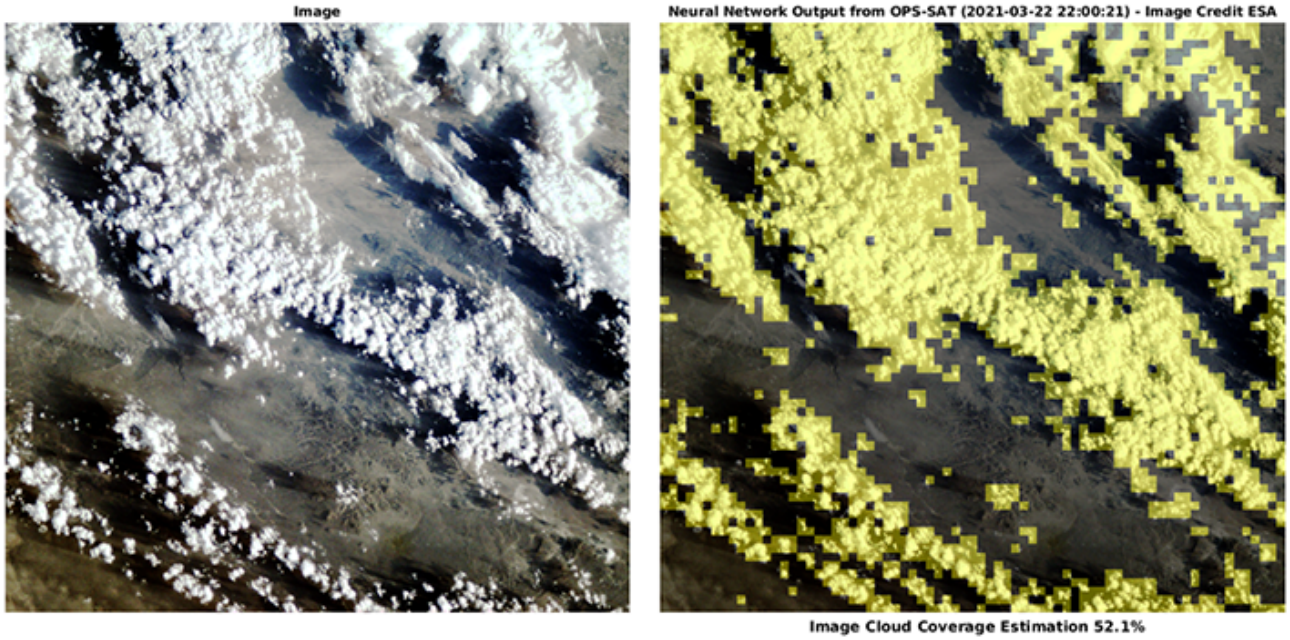


Figure 1: Illustration of the CIAR cloud segmentation task. On the left, the original image taken by OPS-SAT. On the right, the resulting segmentation map, with cloudy patches in yellow. Source: [15]

core ARM Cortex-A9 CPU, 1GB of on chip DD3 memory and 8GB of additional external memory. The board uses 5W in average.

- Optical Camera: a BST IMS-100 camera designed specifically for space applications. At an altitude of 600km, the camera has a field of view of $135 \times 105 kms$, with a resolution of 53m.
- Fine Attitude Control and Determination System (ACDS): a system composed of sun sensors and gyroscopes to measure and control the attitude of the satellite.
- Optical Receiver: a laser communication receiver, embedded for the first time in a CubeSat.
- Software Defined Radio (SDR) front-end: a radio communication system where hardware components (amplifiers, filters, mixers...) have been replaced by software elements. The SDR is used to measure the level of interference in various communication experiments.
- GPS for satellite localization.
- S-Band, X-Band and UHF antennas for communication with the ground base.

APPENDIX 2: Custom SoC platform

In order to test and deploy PADS on FPGA, a custom SoC platform was designed targeting Xilinx Zynq UltraScale+ (ZUS+) boards. An overview of the SoC architecture is provided in Figure 2. The platform has been developed to validate the architecture in hardware, and serve as a testbench for hardware measurement of power consumption. However, probing power on FPGA devices requires specific knowledge and material that were not immediately available. Therefore, this aspect was let aside in the thesis. However, hardware measurements using the custom SoC architecture should take place in further works. Moreover, the platform will be used in other projects at the laboratory and support other FPGA accelerators in further works.

The SoC involves 3 distinct elements:

- The embedded CPU (ARM Cortex A53 MPCore) of the ZUS+ SoC
- The embedded Memory (RAM DDR) of the ZUS+ SoC
- The embedded FPGA (Programmable Logic) of the ZUS+ SoC

The CPU is used to control and exchange data with the Neuromorphic Accelerator synthesized on the FPGA. The FPGA and the CPU are connected through an AXI bus. In the following subsections, each element of the platform is described in details.

.1 Programmable Logic modules

PADS is integrated in the system as an AXI Stream slave peripheral. In doing so, an AXI Stream slave port is added to the IP. Input samples are streamed to PADS trough the AXI Stream interface. Moreover, the AXI Stream slave port is customized for the architecture. First, the data is arranged in a pixel by pixel stream as required by the GenCell (3.1.1). Second, the interface manages PADS control signals and ensures synchronization. The interface also manages the *stop_network*, which triggers a reset signal for PADS.

Additional Xilinx built-in modules are added to the design. For instance, GPIO and AXI Direct Memory Access (DMA) cores are used. The GPIO core provide an interface between external signals of PADS (*o_class* and *stop_network*) and the memory-mapped AXI bus. It should be noted that the *stop_network* GPIO is configured to enable CPU interruption. The AXI DMA core provides a direct interface between the DDR memory and AXI Stream slave peripherals through the memory-mapped AXI bus. The AXI DMA is used to transfer the input samples stored in the DDR to PADS.

This design is configured in Vivado Design Suite, synthesized and implemented for ZCU102 board (ZUS+ family). The Vivado block design synthesized on the FPGA is available in Figure 3. This figure features four distinct areas. The Processing System is the CPU of the board. The Interconnects area contains auto-generated modules for AXI memory-mapping. The Interfaces contains the GPIO and AXI DMA modules. Finally, the PADS area contains the neuromorphic accelerator.

The SoC DDR is mapped on the AXI bus memory-map. The FPGA modules are also mapped on the AXI memory map. DDR and FPGA modules can thus be accessed using those addresses through the AXI bus. The CPU communicates with the accelerator through by this mean.

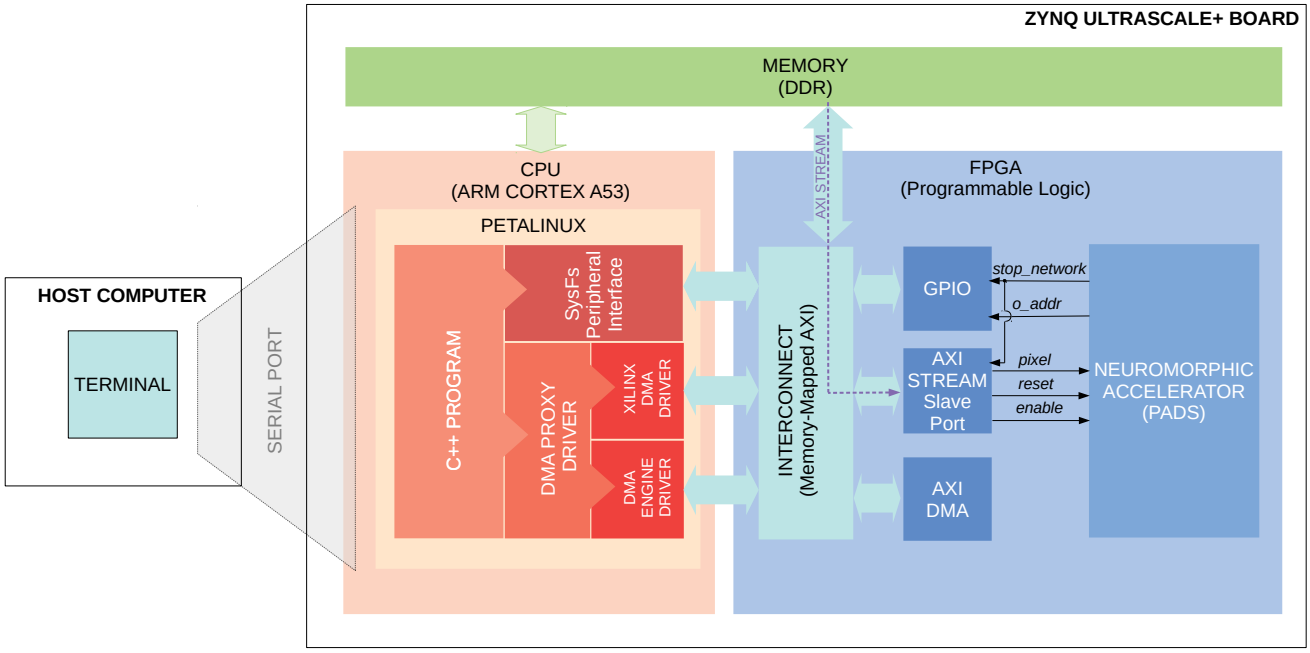


Figure 2: Custom FPGA platform for neuromorphic accelerator deployment. The software stack is in shades of red, and the hardware stack is in shades of blue.

.2 CPU & Embedded Linux

A Petalinux distribution [127] is deployed on the CPU of the Zynq UltraScale+ SoC. The distribution is configured specifically for the platform using a hardware specification file. This file is generated after FPGA synthesis and contains specifications on the hardware design. The Kernel is also configured to enable DMA Drivers and SysFs virtual file-system. The AXI memory-map (containing DDR and FPGA peripheral addresses) is also specified in the device-tree of the Kernel. The Kernel is cross-compiled targeting the ARM-A53 of the SoC.

A C++ program is used to control the SNN accelerator. The software runs on the ARM-A53 core. It is used for data transfer to and from the accelerator. It also manages the control signals (enable, start, stop...)

- Step 1: Initialize AXI DMA and GPIO interfaces.
- Step 2: Write the input sample in the DDR.
- Step 3: Transfer data from DDR to PADS using the AXI DMA.
- Step 4: Wait for the *stop_network* to raise an interruption.
- Step 5: Retrieve the prediction on the GPIO connected to *o_class*.
- Step 6: Wait a few milliseconds and go back to step 2 for a new sample.

This program uses two medium of communication with the peripherals. First, the AXI DMA is accessed through two layers of drivers. At the bottom of the stack, a DMA Engine Driver and a Xilinx DMA Driver enable to control the DMA with a kernel-level API. Additionally, a DMA Proxy Driver is used as an overlay on top of those two drivers. The DMA Proxy Driver enables

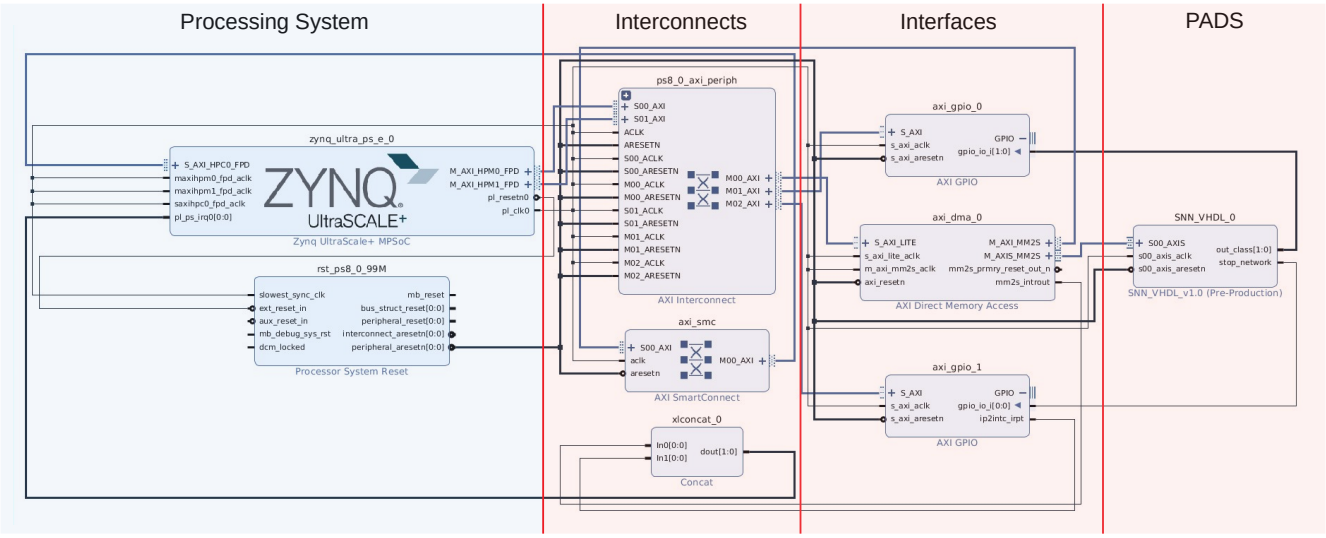


Figure 3: Vivado block design of the FPGA part of the SNN deployment platform.

access to the kernel-level DMA API directly from Linux user-space. This DMA Proxy Driver thus drastically simplifies development. The C++ Program uses the Proxy DMA Driver API manages data transfer between PADS and DDR.

Second, GPIO interfaces are configured and used through a SysFs virtual file-system interface. This interface enables a mapping between the kernel virtual addresses and the AXI bus physical addresses. GPIO addresses are accessed, configured, read and written through the SysFs. It is thus used to access *o_addr* and *stop_network* signals through GPIOs. The SysFs also enables to use the *stop_network* signal as an interruption. Indeed, the C++ program waits for this interruption before starting the next sample. The C++ program is cross-compiled for the custom Petalinux distribution. The Linux image, boot files and compiled C++ application are stored in an SD card alongside data samples. The ZCU102 board is configured in SD mode, thus Petalinux boots automatically when the board is started.

APPENDIX 3: Additional figures

Figure 4: Estimations on Flip-FLop usage for OPS-SAT and Spoken Digits associated CNNs

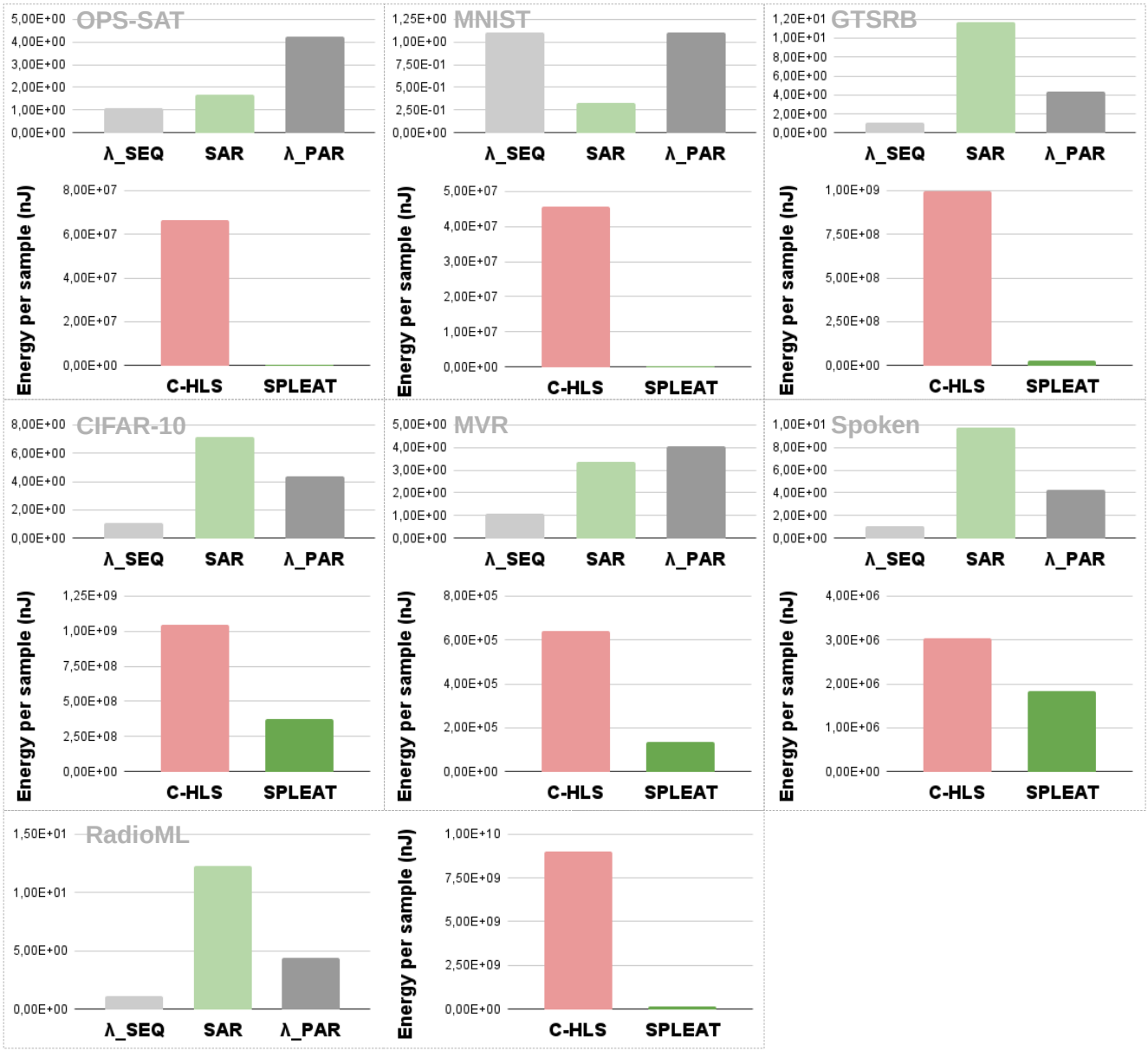


Figure 5: Validation of the SAR metric & λ energy model for SPLEAT versus C-HLS on the benchmark of datasets for full-networks. Top: SAR vs λ_{SEQ}, Bottom: SPLEAT vs C-HLS energy consumption.

APPENDIX 4: Measurements

Table 1: LUT, FF, Block RAM and DSP occupation measures for Convolution layers at design-space measurement points. Measures obtained after hardware synthesis on Xilinx Vivado Design Suite, targeting Xilinx Zedboard

	Look Up Tables			Flip Flops			Block RAM			DSP		
#K - INSIZE	SPLEAT	ANN SEQ.	ANN PAR.	SPLEAT	ANN SEQ.	ANN PAR.	SPLEAT	ANN SEQ.	ANN PAR.	SPLEAT	ANN SEQ.	ANN PAR.
1 - 5	29	156	290,5	19	148	12374,5	0	0	0	0	1	10
1 - 10	88	294	581	68	240	24749	0	0	0	0	1	10
1 - 25	121	233	1057	106	301	1811	0	1	0	0	1	19
1 - 50	146	259	1566	117	338	2985	2	4	0	2	1	20
10 - 5	48	155	4083	35	152	2767	0	0	0	0	1	10
10 - 10	106	295	8166	100	250	5534	0	0	0	0	1	10
10 - 25	143	232	9082	127	301	10509	4	1	0	2	1	8
10 - 50	176	247	10476	152	340	14946	17	4	0	2	2	10
25 - 5	48	158	9135	41	159	6291,5	0	0	0	0	1	10
25 -10	107	296	18270	107	255	12583	0	0	0	0	1	10
25 - 25	145	231	20080	136	301	24014	8	1	0	2	2	10
25 - 50	182	242	24280	213	342	34095	34	4	0	2	2	10
50 - 5	102	152	16207,5	87	164	11765	0	0,5	0	0	1	10
50-10	166	293	32415	156	264	23530	1	0,5	0	0	1	10
50-25	214	222	36598	188	313	45978	17	1,5	0	2	2	10
50-50	320	235	32995	251	351	24637	68	4,5	0	2	2	10
128-5	107	155	32482	96	176	26178	0	1	0	0	1	10
128-8	141	287	64964	134	268	52356	1	1	0	0	1	10
128-16	193	291	57994	193	273	49386	17	1	0	0	1	10

Table 2: LUT, FF, Block RAM and DSP occupation measures for Pooling layers at design-space measurement points. Measures obtained after hardware synthesis on Xilinx Vivado Design Suite, targeting Xilinx Zedboard

	Look Up Tables (#)			Flip Flops (#)			Block RAM (#)			DSP (#)		
#K - IN_WIDTH	SPLEAT	ANN SEQ.	ANN PAR.	SPLEAT	ANN SEQ.	ANN PAR.	SPLEAT	ANN SEQ.	ANN PAR.	SPLEAT	ANN SEQ.	ANN PAR.
1 - 5	107.0	73	116	90.0	80	241	0.0	0	0	0.0	0	0
1 - 10	171.0	85	117	136.0	102	242	0.0	0	0	0.0	0	0
1 - 25	420.0	100	234	157.0	123	513	0.0	0	0	0.0	0	0
1 - 50	1260.0	114	387	215.0	143	909	0.0	0	0	0.0	0	0
10 - 5	182.0	78	885	124.0	86	2176	0.0	0	0	0.0	0	0
10 - 10	559.0	83	966	167.0	104	2177	0.0	0	0	0.0	0	0
10 - 25	2587.0	85	1809	246.0	115	4788	0.0	0	0	2.0	0	0
10 - 50	10606.0	89	3292	615.0	127	8658	0.0	0	0	2.0	1	0
25 - 5	305.0	83	2164	134.0	93	5401	0.0	0	0	0.0	0	0
25 -10	1248.0	82	2163	198.0	106	5402	0.0	0	0	1.0	0	0
25 - 25	6236.0	81	4455	381.0	119	11913	0.0	0	0	3.0	1	0
25 - 50	28199.0	84	9058	1045.0	128	21582	0.0	0	0	3.0	2	0
50 - 5	494.0	89	4291	144.0	100	10778	0.0	0	0	0.0	0	0
50-10	2278.0	87	4302	215.0	117	10780	0.0	0	0	3.0	0	0
50-25	12146.0	88	8827	648.0	126	23780	0.0	0	0	3.0	1	0
50-50	65196.0	83	11233	1356.0	136	27552	0.0	0	0	3.0	2	0
128-16	13715.0	88	15935	1016.0	123	40104	0.0	0	0	0.0	1	0

Table 3: LUT, FF, Block RAM and DSP occupation results for Fully-Connected layers at design-space measurement points. Measures obtained after hardware synthesis on Xilinx Vivado Design Suite, targeting Xilinx Zedboard

	Look Up Tables (#)				Flip Flops (#)				Block RAM (#)				DSP (#)			
#IN - #OUT	**SPLEAT**	**PADS**	**ANN SEQ.**	**ANN PAR.**	**SPLEAT**	**PADS**	**ANN SEQ.**	**ANN PAR.**	**SPLEAT**	**PADS**	**ANN SEQ.**	**ANN PAR.**	**SPLEAT**	**PADS**	**ANN SEQ.**	**ANN PAR.**
10-1	43	136	41	127	20	140	29	207	0	0	0	0	0	0	1	5
10-10	43	1251	46	973	20	1109	34	1676	0	0	0	0	0	0	1	50
10-50	76	6276	48	5417	39	5198	43	8615	0	0	0	0	1	0	1	192
10-100	47	9279	50	16486	29	6205	47	18647	0	0	0	0	0	0	1	220
10-250	45	31398	47	24673	26	25570	52	40767	0	0	1	0	0	0	1	220
50-1	72	467	46	367	35	555	38	452	0	0	0	0	1	0	1	14
50-10	69	4216	51	3584	41	4636	50	4161	0	0	0	0	0	0	1	134
50-50	68	21432	45	31349	44	21635	64	25306	0	0	1	0	1	0	1	220
50-100	77	43682	46	56926	40	43247	69	47932	2	0	2	0	1	0	1	220
50-250	78	111381	46	93149	41	111815	71	100092	1	0	4,5	0	1	0	1	220
100-1	75	824	49	867	38	1036	48	828	0	0	0	0	1	0	1	25
100-10	43	7702	52	9198	20	8596	52	8089	0	0	0	0	0	0	1	220
100-50	79	41505	46	67460	42	41445	66	47814	2	0	2	0	1	0	1	220
100-100	76	83034	47	131657	39	82358	71	93282	1	0	3,5	0	1	0	1	220
100-250	73	216270	47	187192	36	220899	78	182121	0	0	7	0	1	0	1	220
250-1	78	2060	53	2803	41	2532	59	2617	1	0	0	0	1	0	1	66
250-10	43	17739	48	44119	20	20731	61	31081	0	0	1	0	0	0	1	220
250-50	74	96902	51	216015	37	98998	70	147959	1	0	4	0	1	0	1	220
250-100	75	196948	52	361862	38	198768	74	265192	0	0	7	0	1	0	1	220
250-250	72	547792	48	610794	35	628317	82	545554	0	0	14	0	1	0	1	220
800-120	66	212369	61	2363622	43	238607	82	1867470	0	0	27	0	1	0	1	220

Table 4: Power measures for Fully-Connected layers at design-space measurement points. Measures obtained after hardware synthesis and simulation on Xilinx Vivado Design Suite targeting Xilinx Zedboard

	Power (mW)			
#IN - #OUT	**SPLEAT**	**PADS**	**ANN SEQ,**	**ANN PAR,**
10-1	**4**	5	2	10
10-10	**4**	29	2	64
10-50	**5**	108	2	243
10-100	**4**	154	2	497
10-250	**4**	467	2	749
50-1	**5**	11	2	16
50-10	**6**	65	2	105
50-50	**4**	279	2	486
50-100	**8**	535	2	826
50-250	**6**	1259	2	1283
100-1	**5**	22	2	24
100-10	**4**	94	2	155
100-50	**8**	405	2	682
100-100	**6**	732	2	1218
100-250	**5**	1887	2	1792
250-1	**6**	32	2	49
250-10	**4**	164	2	349
250-50	**6**	679	2	1318
250-100	**5**	1338	2	2216
250-250	**5**		2	
800-120	**4**	1174	2	3000

Table 5: Power measures for Convolution layers at design-space measurement points. Measures obtained after hardware synthesis and simulation on Xilinx Vivado Design Suite targeting Xilinx Zedboard

	Power (mW)		
#K - INSIZE	**SPLEAT**	**ANN SEQ.**	**ANN PAR.**
1 - 5	2	4	11
1 - 10	4	5	22
1 - 25	4	4	31
1 - 50	8	5	39
10 - 5	3	4	48
10 - 10	5	5	96
10 - 25	19	4	103
10 - 50	066	5	97
25 - 5	3	4	96
25 -10	5	4	192
25 - 25	36	4	180
25 - 50	138	5	167
50 - 5	5	4	155
50-10	8	5	311
50-25	71	6	274
50-50	144	7	541
128-5	5	4	251
128-8	8	5	503
128-16	73	5	525

Table 6: Power measures for Pooling layers at design-space measurement points. Measures obtained after hardware synthesis and simulation on Xilinx Vivado Design Suite targeting Xilinx Zedboard

	Power (mW)		
#K - IN_WIDTH	**SPLEAT**	**ANN SEQ.**	**ANN PAR.**
1 - 5	0.003	0,01	0,006
1 - 10	0.003	0,01	0,004
1 - 25	0.004	0,01	0,01
1 - 50	0.011	0,01	0,014
10 - 5	0.003	0,01	0,038
10 - 10	0.005	0,01	0,024
10 - 25	0.014	0,01	0,069
10 - 50	0.03	0,01	0,098
25 - 5	0.004	0,01	0,08
25 -10	0.011	0,01	0,045
25 - 25	0.021	0,01	0,156
25 - 50	0.059	0,01	0,23
50 - 5	0.004	0,01	0,147
50-10	0.013	0,01	0,079
50-25	0.033	0,01	0,296
50-50	0.067	0,01	0,358
128-16	0.039	0,01	0,535

Table 7: Duration results for Fully-Connected layers at design-space measurement points. Measures obtained by calculation and validated using post-synthesis simulation.

	Duration (ns)			
#IN - #OUT	**SPLEAT**	**PADS**	**ANN SEQ,**	**ANN PAR,**
10-1	**40**	60	340	70
10-10	**130**	60	3310	70
10-50	**530**	60	16510	70
10-100	**1030**	60	33010	70
10-250	**2530**	60	82510	70
50-1	**40**	80	1540	90
50-10	**130**	80	15310	90
50-50	**530**	80	76510	90
50-100	**1030**	80	153010	90
50-250	**2530**	80	382510	90
100-1	**40**	90	3040	100
100-10	**130**	90	30310	100
100-50	**530**	90	151510	100
100-100	**1030**	90	303010	100
100-250	**2530**	90	757510	100
250-1	**40**	100	7540	110
250-10	**130**	100	75310	110
250-50	**530**	100	376510	110
250-100	**1030**	100	753010	110
250-250	**2530**	100	1882510	110
800-120	**1230**	120	3333910	130

Table 8: Duration results for Convolution layers at design-space measurement points. Measures obtained by calculation and validated using post-synthesis simulation.

	Duration (ns)		
#K - INSIZE	**SPLEAT**	**ANN SEQ.**	**ANN PAR.**
1 - 5	290	5020	34
1 - 10	290	163820	109
1 - 25	290	1979870	685
1 - 50	290	9476620	2509
10 - 5	2540	9420	34
10 - 10	2540	325420	109
10 - 25	2540	3946720	685
10 - 50	2540	18902220	2509
25 - 5	6290	22620	34
25 -10	6290	810220	109
25 - 25	6290	9847270	685
25 - 50	6290	47179020	2509
50 - 5	12540	44620	34
50-10	12540	1618220	109
50-25	12540	19681520	685
50-50	12540	94307020	2509
128-5	32040	113260	34
128-8	32040	2252140	73
128-16	32040	25806500	265

Table 9: Duration results for Pooling layers at design-space measurement points. Measures obtained by calculation and validated using post-synthesis simulation.

	Duration (ns)		
#K - INSIZE	**SPLEAT**	**ANN SEQ.**	**ANN PAR.**
1 - 5	50	3110	50
1 - 10	50	27860	1040
1 - 25	50	102110	6290
1 - 50	50	440110	25040
10 - 5	50	6210	50
10 - 10	50	60971	1040
10 - 25	50	226583	6290
10 - 50	50	977317	25040
25 - 5	50	15510	50
25 -10	50	157260	1040
25 - 25	50	582510	6290
25 - 50	50	2513010	25040
50 - 5	50	31010	50
50-10	50	541510	1040
50-25	50	1165010	6290
50-50	50	5026010	25040
128-16	50	1950470	2600

APPENDIX 5: Raw Estimations

SPLEAT

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
MNIST	**C1**	258	289	4	2	0.023	439044	10098
	P1	1626	221	0	2	0.009	852340	7671
	C2	228	265	1	1	0.010	1384758	13847
	P2	696	174	0	1	0.006	325090	1950
	FC1	123	119	7	1	0.018	272561	4906
	FC2	106	91	0	0	0.005	25199	125
	TOTAL	3035	1157	12	5	0.071	1384758	444507

PADS

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
MNIST	**C1**							
	P1							
	C2							
	P2							
	FC1	164934	166842	0	0	0.896	18300.0	16396
	FC2	6587	7329	0	0	0.062	23104.0	1432
	TOTAL	171520	174171	0	0	0.958	23104	27909

VGT

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
MNIST	**C1**	4396	3824	0	10	0.098	6680	654
	P1	1099	2871	0	0	0.098	6390	626
	C2	72532	50272	0	60	0.962	4010	3857
	P2	1442	3466	0	0	0.048	3090	148
	FC1	315191	227678	0	220	1.928	13290	25623
	FC2	7402	6833	0	193	0.138	5770	796
	TOTAL	402060	294940	0	483	3.220	15780	54756

C-HLS

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
MNIST	**C1**	233	286	4	1	0.014	125036600	1750512.4
	P1	91	118	1	0	0.016	977800	15644.8
	C2	303	262	1	1	0.011	60130800	661438.8
	P2	81	98	0	0	0.016	445800	7132.8
	FC1	60	78	10	1	0.019	6325300	120180.7
	FC2	45	54	1	1	0.017	255100	4336.7
	TOTAL	811	893	17	4	0.095	193171400	66644133

SPLEAT

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
OPSSAT	**C1**	246	265	3	1	0.017	1024304.0	17413
	P1	907	192	0	1	0.006	296333.0	1777
	C2	202	210	1	1	0.009	315095.0	2835
	P2	210	146	0	0	0.003	591702.2	1775
	FC1	106	91	0	0	0.005	58480.0	292
	FC2	65	51	0	0	0.002	38455.0	76
	TOTAL	1733	952	4	3	0.042	1024304	299096

PADS

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
OPSSAT	**C1**							
	P1							
	C2							
	P2							
	FC1	6308	7012	0	0	0.062	46076	2856
	FC2	260	248	0	0	0.004	38395	153
	TOTAL	6568	7260	0	0	0.066	46076	14560

VGT

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
OPSSAT	**C1**	6737	7047	0	30	0.19	6680	1269.2
	P1	571	1446	0	0	0.022	6390	140.58
	C2	11500	8702	0	30	0.202	4020	812.04
	P2	492	1100	0	0	0.014	3100	43.4
	FC1	6953	6518	0	185	0.135	1450	195.75
	FC2	221	371	0	10	0.016	350	5.6
	TOTAL	26472	25183	0	255	0.58	9850	8175.5

C-HLS

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
OPSSAT	**C1**	207	266	3	1	0.012	113235399	1358824.788
	P1	96	120	1	0	0.016	971600	15545.6
	C2	276	232	1	1	0.011	19399300	213392.3
	P2	83	96	0	0	0.016	377600	6041.6
	FC1	45	54	0	1	0.017	33100	562.7
	FC2	22	29	0	1	0.016	6433	102.928
	TOTAL	727	796	5	4	0.09	134023432	45567966.88

SPLEAT

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
GTSRB	**C1**	294	366	18	2	0.081	32616662	2641949.622
	P1	10543	552	0	3	0.028	13090140	366523.92
	C2	240	295	5	1	0.026	59256398	1540666.348
	P2	1551	218	0	1	0.011	5332540	58657.94
	FC1	148	128	37	1	0.035	4618640	161652.4
	FC2	122	118	4	1	0.013	886004	11518.052
	FC3	115	106	2	1	0.008	332371	2658.968
	TOTAL	13011	1780	65	10	0.202	59256398	26783891.9

PADS

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
GTSRB	**C1**							
	P1							
	C2							
	P2							
	FC1	580849	741511	0	0	3.443	139830	481434.69
	FC2	84934	84788	0	0	0.472	144558	68231.376
	FC3	32069	32131	0	0	0.202	196552	39703.504
	TOTAL	697851	858429	0	0	4.117	196552	858342.584

VGT

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
GTSRB	**C1**	56740	45131	0	30	0.955	8920	8518.6
	P1	6287	16397	0	0	0.204	8590	1752.36
	C2	671751	489375	0	320	8.884	5870	52149.08
	P2	2762	6908	0	0	0.098	4830	473.34
	FC1	2606894	1967153	0	220	10.667	19050	203206.35
	FC2	141808	101654	0	221	1.179	19680	23202.72
	FC3	51046	36912	0	205	0.552	14230	7854.96
	TOTAL	3537286	2663527	0	995	22.542	33160	755782.72

C-HLS

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
GTSRB	**C1**	270	318	15	2	0.02	691612500	13832250
	P1	86	123	4	2	0.02	9772700	195454
	C2	293	273	4	2	0.013	1873068800	24349894.4
	P2	84	110	1	0	0.016	1616000	25856
	FC1	67	80	48	1	0.025	28836100	720902.5
	FC2	53	72	6	1	0.018	3145300	56615.4
	FC3	47	63	3	1	0.017	255100	4336.7
	TOTAL	897	1036	77	9	0.132	2608306500	996373083

SPLEAT

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
CIFAR-10	**C1**	294	366	18	2	0.081	141491930	11460846.33
	P1	10543	552	0	3	0.028	38334740	1073372.72
	C2	247	318	6	1	0.03	742796350	22283890.5
	P2	2917	250	0	3	0.014	16405740	229680.36
	C3	159	189	1	0	0.006	898385570	5390313.42
	FC1	107	92	1	0	0.004	218635	874.54
	TOTAL	14265	1765	26	9	0.163	898385570	371033240.4

PADS

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
CIFAR-10	**C1**							
	P1							
	C2							
	P2							
	C3							
	FC1	9576	10862	0	0	0.077	218585	16831.045
	TOTAL	9576	10862	0	0	0.077	218585	71477.295

VGT

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
CIFAR-10	**C1**	56740	45131	0	30	0.955	10290	9826.95
	P1	6287	16397	0	0	0.204	8940	1823.76
	C2	1172139	901745	0	320	12.663	8670	109788.21
	P2	5494	13791	0	0	0.165	5890	971.85
	C3	2070432	1624128	0	320	17.504	4920	86119.68
	FC1	15717	12381	0	220	0.191	1850	353.35
	TOTAL	3326807	2613572	0	890	31.683	12350	394372.55

C-HLS

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
CIFAR-10	**C1**	270	318	15	2	0.02	691612500	13832250
	P1	86	123	4	2	0.02	9772700	195454
	C2	307	282	5	2	0.015	2204025600	33060384
	P2	94	120	1	0	0.017	6066900	103137.3
	C3	198	176	2	1	0.011	72486400	797350.4
	FC1	46	56	1	1	0.016	387100	6193.6
	TOTAL	999	1073	26	8	0.1	2984351200	1044522920

SPLEAT

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
MVR	**FC1**	127	130	4	1	0.013	513805	6679.465
	FC2	60	51	0	0	0.003	24460	73.38
	TOTAL	186	181	4	1	0.016	513805	136672.13

PADS

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
MVR	**FC1**	96686	96782	0	0	1.053	3221	3391.713
	FC2	2413	2874	0	0	0.036	3239	116.604
	TOTAL	99098	99655	0	0	1.089	3239	4337.021

VGT

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
MVR	**FC1**	96021	91776	0	220	1.209	31850	38506.65
	FC2	3084	2828	0	74	0.055	31850	1751.75
	TOTAL	99105	94603	0	294	1.264	2780	4208.92

C-HLS

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
MVR	**FC1**	55	72	4	1	0.018	1830100	32941.8
	FC2	41	50	1	1	0.017	423766	7204.022
	TOTAL	95	122	5	2	0.035	2253866	642351.81

SPLEAT

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
SPOKEN	**C1**	233	249	1	1	0.011	196172	2157.892
	P1	873	189	0	1	0.006	2947540	17685.24
	C2	215	228	2	1	0.014	377495	5284.93
	P2	1174	197	0	1	0.007	5568490	38979.43
	C3	199	205	1	1	0.008	604133	4833.064
	P3	1174	197	0	1	0.004	3049290	12197.16
	FC1	125	124	10	1	0.023	4058808	93352.584
	FC2	106	91	0	0	0.005	55539	277.695
	TOTAL	3420	1450	14	7	0.078	5568490	1826464.72

PADS

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
SPOKEN	**C1**							
	P1							
	C2							
	P2							
	C3							
	P3							
	FC1	216639	233426	0	0	1.402	29431	41262.262
	FC2	7702	8596	0	0	0.071	38369	2724.199
	TOTAL	224341	242022	0	0	1.473	38369	66109.787

VGT

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
SPOKEN	**C1**	2225	2208	0	10	0.059	5510	325.09
	P1	556	1410	0	0	0.022	4810	105.82
	C2	10785	8852	0	30	0.228	6270	1429.56
	P2	832	2147	0	0	0.032	5410	173.12
	C3	19463	14441	0	50	0.326	5700	1858.2
	P3	832	2147	0	0	0.032	2790	89.28
	FC1	562019	409155	0	220	3.075	15850	48738.75
	FC2	9198	8089	0	220	0.155	5770	894.35
	TOTAL	605626	447577	0	530	3.921	15070	62856.97

C-HLS

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
SPOKEN	**C1**	212	256	1	1	0.012	19302733	231632.796
	P1	94	118	1	0	0.016	872600	13961.6
	C2	262	247	2	1	0.013	48369700	628806.1
	P2	85	108	1	0	0.016	575600	9209.6
	C3	279	229	1	1	0.011	20261166	222872.826
	P3	85	108	1	0	0.016	328100	5249.6
	FC1	62	80	13	1	0.02	8880100	177602
	FC2	45	54	0	1	0.017	303100	5152.7
	TOTAL	1122	1192	20	5	0.123	98893099	3040962.794

SPLEAT

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
RADIOML	**C1**	294	366	18	2	0.081	19938770	1615040.37
	P1	14058	659	0	3	0.035	260275990	9109659.65
	C2	288	372	23	2	0.099	196932750	19496342.25
	P2	12579	685	0	3	0.034	24896490	846480.66
	C3	265	350	17	1	0.075	50042510	3753188.25
	P3	12579	685	0	3	0.007	3595730	25170.11
	FC1							
	FC2	126	126	10	1	0.023	1366975	31440.425
	FC3	111	98	2	1	0.005	325310	1626.55
	TOTAL	28892	2849	68	16	0.359	260275990	158508077.9

PADS

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
RADIOML	**C1**							
	P1							
	C2							
	P2							
	C3							
	P3							
	FC1							
	FC2	265065	283572	0	0	1.249	338255	422480.495
	FC3	29874	30837	0	0	0.16	324715	51954.4
	TOTAL	294939	314408	0	0	1.409	338255	561165.045

VGT

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
RADIOML	**C1**	56740	45131	0	30	0.955	10590	10113.45
	P1	6888	17943	0	0	0.215	10570	2272.55
	C2	1148879	901156	0	320	15.062	13230	199270.26
	P2	10317	26535	0	0	0.337	12650	4263.05
	C3	4165095	3362637	0	640	44.595	13030	581072.85
	P3	10317	26535	0	0	0.337	8410	2834.17
	FC1							
	FC2	435017	336722	0	220	2.833	43450	123093.85
	FC3	57320	40687	0	220	0.458	14570	6673.06
	TOTAL	5881084	4732955	0	1430	64.49	27820	1801066.8

C-HLS

		LUT (#)	FF (#)	RAM (#)	DSP (#)	Dyn. Pow. (W)	Time (ns)	Energy (nJ)
RADIOML	**C1**	270	318	15	2	0.02	691612500	13832250
	P1	88	124	4	2	0.02	12861500	257230
	C2	283	309	14	2	0.02	6904563200	138091264
	P2	96	127	5	1	0.02	13211100	264222
	C3	320	297	15	2	0.02	14482726400	289654528
	P3	96	127	5	1	0.02	19504700	390094
	FC1							
	FC2	60	79	14	1	0.021	10088500	211858.5
	FC3	48	60	2	1	0.017	1147299	19504.083
	TOTAL	1248	1420	66	12	0.158	22135715199	9031371801