Efficiency analysis of Artificial vs Spiking Neural Networks on FPGAs

Zhuoer Li, Edgar Lemaire, Nassim Abderrahmane, Sébastien Bilavarn, Benoit Miramond

Efficiency Analysis of Artificial vs. Spiking Neural Networks on FPGAs

ZHUOER LI[1], EDGAR LEMAIRE[2], NASSIM ABDERRAHMANE[3], SÉBASTIEN BILAVARN[1], AND BENOÎT MIRAMOND[1]

[1] *LEAT, CNRS UMR7248, Université Côte d'Azur, France*
[2] *High-Performance Computing Lab, Thales Research & Technology, Palaiseau, France*
[3] *IRT Saint Exupéry, Sophia-Antipolis, France*

Artificial neural networks (ANNs) incur huge costs in terms of processing power, memory performance, and energy consumption, where in comparison an average human brain operates within a power budget of nearly 20 W. Brain-inspired computing such as Spiking Neural Networks (SNNs) are thus expected to improve efficiency to an unprecedented extent. But apart from the spike coding aspects currently addressed by numerous investigations, research also needs to find solutions for the practical design of future neuromorphic hardware ensuring very low power processing. This paper investigates these questions with a pragmatic comparison of deep Convolutional Neural Networks (CNNs) and their equivalent SNNs based on the implementation and measurement of a set of CNN image classification benchmarks on FPGA devices. Results show that SNNs are clearly less energy efficient than their equivalent CNNs in the general case, further indicating that, on top of ongoing progress in spike modeling theory (e.g. spike encoding, learning), neuromorphic accelerators also have to address important issues in the reality of RTL development and silicon implementation, among which sparsity versus static and idle power consumption, ability to support large levels of parallelism, memory performance, scalability, spiking convolutions.

1. INTRODUCTION

Artificial Intelligence (AI) has grown in popularity for a number of embedded applications ranging from speech and image processing to autonomous navigation, robotics, or IoT. But processing AI algorithms with limited hardware resources raises a number of challenges in terms of complexity, memory, latency, and even more concerns with energy efficiency. The resulting Edge computing effort, i.e. the ability to process data at the edge locally rather than in the cloud, meets the still unavoidable problem of power efficiency of SoCs, with the deepening environmental impact of digital technologies and their growing share of greenhouse gas emissions (5G, Video on Demand, Cloud services, blockchain, AI, ...) [1].

In the more specific context of Deep Neural Networks (DNNs), FPGA technology has long been regarded as a very promising acceleration device for embedded computing platforms because it combines good power/performance trade-off with flexibility. In addition, Spiking Neural Networks (SNNs) constitute a great hope over classical formal deep learning methods (Convolutional Neural Networks, CNNs) in terms of processing efficiency. However, it is still unclear if SNN hardware is truly better due to difficulty and lack of actual implementations and therefore comparison. To address this fundamental efficiency issue, the following study investigates a practical comparison of CNN and SNN implementations on FPGAs based on High-Level Synthesis (HLS).

The outline of the paper is as follows. We first review existing works in the field of DNNs, AI processors, and accelerators, pointing out a variety of energy-efficient hardware/software contributions that can lead to pertinent proposals for further efficiency. In section 3, we introduce the proposed methodology to develop, implement and compare properly CNN and SNN accelerators on FPGA devices. Detailed results using different network topologies and datasets are then analyzed and discussed in section 4. Finally, we present our principal conclusions from the detailed case studies and future directions for research and applications.

2. STATE OF THE ART

Artificial Intelligence (AI) is investigating various directions as more computing is moving to the Edge. Offloading AI to resource-limited devices requires not only to complete more computations per second but also to perform at much less power. The extent of this challenge is not limited to a classical hardware design issue, but rather encompasses both hardware and software innovative solutions in a power-smart approach for higher levels of processing and energy efficiency.

At the algorithmic level, Convolutional Neural Networks (CNNs) have set the standard in deep learning architectures through unmatched performances in image and speech recognition tasks. However, their compute complexity is impaired by an enormous amount of Multiply-Accumulate (MAC) operations and memory accesses. Binarized Neural Networks (BNNs) have therefore been developed on the idea of converting floating-

point weights to binary weights. Simplifying MACs to simple XNOR and Pop-Count operations this way proved to achieve comparable accuracy to CNNs [2], additionally saving storage, calculation, power, and energy consumption. Another promising direction at the algorithmic level is the exploration of "brain-like" biorealistic computational models that more closely mimic natural neural networks. Spiking Neural Networks (SNNs) consider time information, that is, not all neurons are activated in each propagation iteration, but only when their membrane potential reaches a certain value. When a neuron is activated, it generates a signal, which is transmitted to connected neurons, raising or lowering their membrane potential. In SNNs, information processing is event-driven which means that as long as there is little or no recorded information, the SNN should not compute too much and lead to an energy-efficient calculation process. In this regard, SNNs have great promise to improve the performance and efficiency of neural network processing [3].

On the hardware side, AI chip technology is still in its infancy. There is a challenge around the technology hardware that can lead to the best effective trade-off between acceleration, efficiency, scalability, and flexibility for rapidly changing and different types of Machine Learning (ML) algorithms. AI processors rely usually on parallel computing elements sometimes associated with (a) generic CPU(s) to deliver the performance ML requires. This is often the approach adopted by mainstream CPU manufacturers. The Ethos microarchitecture solution proposed by Arm for instance [4] [5] is based on a ML processor that can be configured with up to sixteen Compute Engines (CE). Each CE is composed of a MAC Compute Engine (MCE), a Programmable Layer Engine (PLE), and a configurable amount of SRAM (64 KiB or 256 KiB). The MCE contains the high-efficiency fixed-function MAC units while the PLE contains the flexible programmable vector engine (consisting of a Cortex-M processor). IBM Research [6] proposed a multicore chip made of four AI cores optimized for low precision training (8-bit) and inference (4-bit) with improved data communications. The chip is designed in a state-of-the-art silicon technology node (7 nm EUV-based chip) and is notably one of the first AI chips to include power management.

Regarding SNN processors, the University of Manchester started to work early on a massively parallel computer called SpiNNaker [7]. The basic building block of the architecture is the SpiNNaker Chip Multiprocessor (CMP) which is a custom-designed globally asynchronous locally synchronous (GALS) system with 18 ARM968 processor nodes residing in synchronous islands, surrounded by a lightweight, packet-switched asynchronous communications infrastructure. Every neuron is represented by a discrete core and constitutes a truly event-driven system. Because neurons are emulated by ARM cores, this platform is more intended for human brain simulation and research, rather than for overall power reduction. IBM TrueNorth [8] is more relevant in terms of low power design with a manycore processor network on a chip of 4096 neurosynaptic cores tiled in a 2-D array, each implementing 256 neurons and 64k synapses. A neurosynaptic core is an extremely low power processing unit emulating a neuron that can further connect to any axon of any other neurosynaptic core (including itself). The system is, therefore, able to address large complex network topologies, is natively event-driven, doesn't have a clock, and is very energy-efficient, operating at lower temperatures and power. Another reference example is given by Intel who developed Loihi [9], another event-based computing architecture for SNN computational models. This processor is made of 128 neuromorphic cores, three embedded x86 processor cores, and off-chip communication interfaces. Each neuromorphic core implements 1024 primitive spiking neural units grouped into sets of trees constituting neurons, so it is capable of simulating around 130000 neurons and 130 million synapses in total.

However, the computing nature and power associated with energy constraints for AI processing at the Edge requires highly specialized hardware to be really efficient. This has also led to the development of important research on hardware acceleration for neural networks. AI accelerators have many optimizations that make them suitable for their use-case, such as parallelism, low-precision arithmetic, advanced low-level architecture, etc. Graphic Processing Units (GPUs) were the first acceleration devices used to improve the execution of Deep Learning models. The mathematical basis of neural network and image processing is similar, and parallel tasks involving matrices are suitable for GPU to complete. GPUs can provide high memory bandwidth and throughput, and they are very efficient in floating-point matrix-based operations. In addition, compared with hardware development related to FPGAs or ASICs, GPU programming is a lot easier to manage for AI developers. However, GPU power consumption will also set out continued challenges to assure their effective use in embedded systems [10].

Dedicated AI acceleration is the most promising alternative regarding the processing efficiency of neural networks. Many companies such as Facebook, Amazon, and Google are all designing their own AI ASICs, and there is also much academic research. Eyeriss [11] for example is an accelerator for state-of-the-art deep convolutional neural networks (CNNs) developed by MIT. Like dedicated hardware does for the implementation of any given processing task, Eyeriss exploits specialization and parallelism to achieve drastic latency and energy reductions for deep CNNs. It further optimizes the processing efficiency of the entire system for various CNN shapes by reconfiguring the architecture based on a spatial array of 12x14 processing elements, with a NoC and memory hierarchy designed specifically to reduce expensive data movement such as DRAM accesses. A chip was implemented in 65-nm CMOS, processing the convolutional layers at 35 frames/s for AlexNet at 278 mW, and 0.7 frames/s for VGG-16 at 236 mW with a 200-MHz core clock.

Regarding dedicated accelerators, FPGA technology is also a very attractive option to deal with power issues [12], at an efficiency level slightly below ASICs. However reconfigurable hardware provides additional flexibility allowing to adjust a variety of design parameters (type, topology, parallelism, etc.) together with the reconfiguration ability. Among the two leading FPGA vendors, Xilinx comes up with a Deep Learning Processor Unit (DPU) [13] which is a configurable IP block supporting all CNN topologies, tightly coupled with the main CPU of a Zynq processing system and using a specific DPU instruction set. Intel FPGAs for AI rather consider the architecture aspect with AI Tensor Blocks [14] which are AI optimized blocks tuned for common matrix-matrix and vector-matrix multiplications and INT8 inferencing. In terms of development, it should be noted that FPGAs are becoming of growing interest against other programmable solutions (AI processors, GPUs) with the recent maturity of High-Level Synthesis approaches to automate the mapping of CNNs onto reconfigurable FPGA-based platforms, such as in [15].

Other directions are additionally explored to allow further AI energy efficiency. A promising processor-related area is heterogeneity adaptation and management. For example, Conti et al. [16] addressed a solution for IoT based on a combination

of a low power microcontroller unit (MCU) and heterogeneous programmable accelerator, reaching up to 60x gains in performance and energy efficiency on a diverse set of applications. In this regard, power management is also an important issue that starts to be tackled. For example, Höppner et al. [17] developed a dynamic power management solution for SpiNNaker based on fast dynamic voltage and frequency scaling (DVFS) allowing to reduce the total PE power consumption by 75%, paving the way for further saving potential and new power management strategies specific to AI.

Neural networks are highly constrained by the need of moving large amounts of data (weights and input) from/to memory. This has led to a variety of efforts, on a more technical level, addressing questions related to near-memory computing, in-memory computing, non-volatile memories, and silicon technology. Near-Memory computing wishes to incorporate memory and logic chips in an advanced IC package to bring them closer to the processing cores, achieving high-bandwidth data communication through 2.5D / 3D stacking or increasing the number of cache levels, high-density on-chip storage, and other near-data storage to alleviate memory access latency and high power consumption. For example, [18] designed an efficient near-memory acceleration engine reaching 2.7 times energy efficiency improvement over contemporary GPUs at 4.4 times less silicon area. In-Memory computing seeks to run computer calculations entirely in computer memory, thereby reducing the frequency of processor access to memory. The current realization of in-memory computing mainly focuses on two types of memory: volatile SRAM or DRAM, non-volatile phase-change memory (PCM), resistive random access memory/memristor, magnetic random access memory, and floating gate device / Flash construction. IBM designed a PCM-based analog AI core that reduces power consumption by 10,000 - 2,000,000 times compared with conventional computers [19]. Finally, advances in silicon technology are also affecting the improvement of energy efficiency. The latest 2nm chip announced by IBM, compared with modern 7nm processors, improves performance by 45% at the same power level and energy by 75% at the same performance level [20].

This great variety of existing approaches for AI processing efficiency calls for additional reflection on further improvements and possible combinations at a more global level. In the following, we aim to address these questions, first at the algorithmic level to better characterize different processing models (CNN, SNN), and second at the NN deployment and execution levels to improve efficiency even more. This approach will be based therefore on experimental investigations focusing on FPGA prototyping, measurement, and analysis. It will address firstly a comparison study of CNNs and SNNs, a fundamental issue to address since, despite a broad consensus on the potential of SNNs, it is still unclear to what extent this is realistic to assume. This effort is made easier thanks to using High-Level Synthesis to generate fully operational CNNs and SNNs on FPGAs. Opportunities for the use of other contributions such as power and heterogeneity management will be also envisaged.

3. COMPARISON METHODOLOGY

A. Benchmarking DNN accelerators

In the following, we introduce the set of CNN and SNN IPs that will be used for comparison and analysis, principally in terms of processing and energy efficiency. High-Level Synthesis (HLS) is thus very beneficial to help to prototype and explore various hardware implementation opportunities (parallelism, memory allocation, FPGA device, etc.). Several network topologies and image classification datasets are also used (MNIST, GTSRB, CIFAR-10) to increase the relevance of the measurement campaigns. The main goal is to compare real CNN accelerators with their spike-coding equivalents on the same deep neural network benchmarks.

HLS is used to produce quickly and more easily various fully functional accelerators on FPGA devices that can operate under the control of an embedded CPU core. A physical SNN IP developed manually in VHDL is also used to assess the quality of HLS results (both in terms of logic resources and latency), especially for spiking accelerators. Therefore, three types of DNN accelerators are used: CNN hardware generated using HLS (CNN HLS), SNN hardware generated with HLS (SNN HLS), and SNN RTL hardware developed in VHDL manually (SNN VHDL). All three globally share the same layer-based architecture of figure 1, which may help to completely design any network topology combining convolution (CONV), max-pooling (POOL), and fully connected (FC) layers. For all HLS implementations (CNN, SNN), intra-layer parallelism is broadly addressed by exploring the potential of loop level optimization and array partitioning in a way to ensure the best level of performance. Thereafter, different measurements are carried out where logic resources, latency, and power characterizations are determined for i) the most sequential (ZedBoard) and ii) the most parallel (ZCU102) implementation of HLS generated accelerators.

B. CNN inference design with HLS

A plain C code was written specifically to allow fast simulation of ConvNets supporting HLS and full hardware/software execution on FPGA platforms. This code is a layer-based implementation allowing the acceleration of 2D convolutions, ReLU, max pooling, and fully connected layers. Other processing functions such as file reading, pre-processing (image) and post-processing (activation functions) are less suited to hardware execution and remain in software. This allows for unlimited combinations of layers to simulate easily most network topologies while keeping them entirely HLS compliant. The approach allows to set up a CNN model with Keras, where the weights and bias values learned are then used for inference in the C code.

The resulting C model can be used further to produce RTL IP blocks that can be synthesized targeting any ASIC or FPGA technology. For each CNN implementation, loop parallelism is addressed manually with the use of HLS pragmas (unroll, pipeline, array partition) in a way to get the best possible performance. For our needs, these accelerators are then integrated with a CPU core and system bus to be quickly executed on programmable SoC FPGAs following Xilinx Vivado, Vivado HLS, and SDSoC methodologies. This approach allows for complete system integration including operating system, graphical display, file system, drivers, and therefore realistic measurements, powerful demos, or application releases that are implemented in the following on ZedBoard and ZCU102 platforms.

C. Spiking accelerators

HLS-generated accelerators for SNNs are developed and used for experimentation and measurement in the first place because they are easier to design and fully prototype on FPGA platforms. A manually designed accelerator in VHDL is used though for

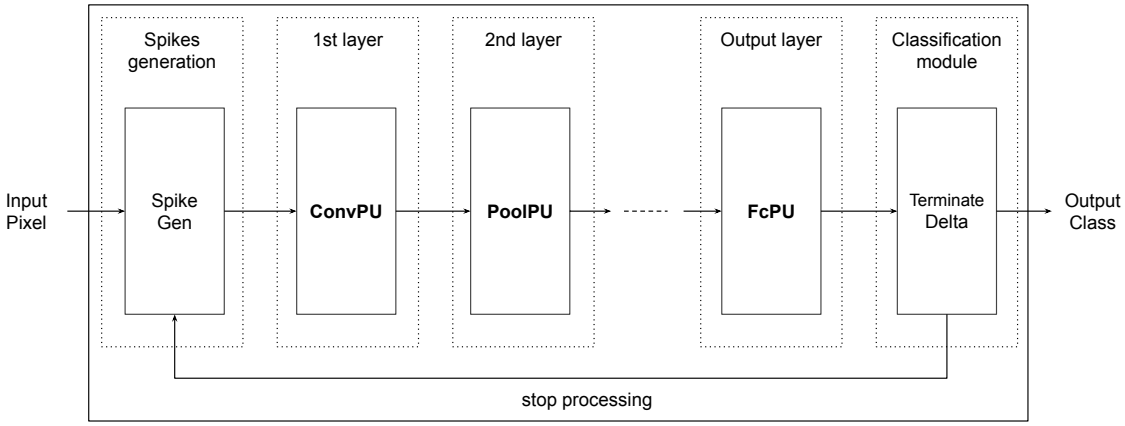

Fig. 1. Overview of underlying spiking hardware architectures

the purpose of assessing the relevance of the HLS IPs on one single SNN topology (MNIST, section 4.2).

However, there is no means in practice to develop SNN accelerators in HLS compliant C or C++ due to the event-based nature of spike processing. Nevertheless, they can be fully specified and simulated with SystemC, and using the Synthesizable Subset defined by the Accellera System Initiative adds full RTL synthesis capability to the simulable models. The resulting simulation/synthesis approach greatly facilitates the full development of SNN IPs by simply aggregating the layers of a NN topology, training the equivalent neural network with Keras, simulating in SystemC, and synthesizing quickly the RTL automatically to finally create a functional hardware/software system controlled by a FPGA processor core [21].

Both manual and HLS SNN accelerators are designed so that each layer of the neural network (convolution, pooling, fully connected) is implemented in the spiking domain. One layer is made of a single neuronal unit following a classical Integrate-and-Fire (IF) model (Fig. 1). This approach, in comparison with the instantiation of a very large number of neurons in parallel within a layer, allows to limit hardware resources and to simplify the architecture. This implies a sequential multiplexed execution of one physical neuron (per layer) where no more than one spike can be processed at a time.

The full spiking accelerators additionally include transcoding (Spike Gen) and classification (Terminate Delta) modules to respectively generate spike trains from the pixels of an input image and finish execution when the winning class is found. Further details and descriptions of these design related concerns, which relates exactly to the hardware architecture model of (Fig. 1), are available in [22]. In the end, both manual and automated SNN IPs are truly event-based neuromorphic processing accelerators, completely synthesizable and functional on FPGA platforms. Adaptations have been made to the original Xilinx integration flow to support designs including SystemC IP blocks. It is therefore possible to run the HLS SNN IPs on Zynq platforms controlled by an ARM CPU, possibly under Linux. This makes it possible to i) further derive the actual inference times from real execution on different platforms and ii) analyze in depth the complete architecture, in particular, to let Xilinx synthesis tools compute low-level power estimates.

4. COMPARISON STUDY

For the sake of comparison, we implement, execute, monitor, and report the execution of four typical deep learning models for both CNN and SNN accelerators. Three different datasets are used for the measurement: MNIST (Modified National Institute of Standards and Technology database, 28x28 black and white handwritten digital numbers), GTSRB (German Traffic Sign Recognition Benchmark, 32x32 color traffic sign pictures), and CIFAR-10 (Canadian Institute For Advanced Research, 32x32 color objects or animal pictures). These datasets are used in four topologies that are defined and trained with Keras to serve as a reference basis and provide weights and biases for CNN and SNN inference.

SNN training is based on CNN supervised learning using back-propagation, i.e. neural network conversion. A CNN model with the same topology is used. The only difference is the use of a linear activation function in the last output layer since SNNs do not need to employ elaborated differentiable neurons (rectified linear or sigmoid) like for CNNs. It should be noted that SNN accuracy is affected by the systematic use of a linear activation instead of a softmax function in the output layer of the corresponding CNN used for training. Because of this, SNN accuracy is actually less than that of the equivalent CNN and the difference tends to grow with the complexity of the topology (GTSRB: 85.20% vs. 91.99%, CIFAR10: 79.20% vs. 88,3%).

For each topology, we implement a CNN and its equivalent SNN on ZedBoard (sequential version) and ZCU102 (parallel version). In the following comparison analysis, we refer primarily to the parallel versions of accelerators (on ZCU102) for performance-related reasons. Sequential implementations (on ZedBoard) will be helpful to analyze performance and parallelism-related issues (section 4.5).

A. Multilayer perceptron

The first topology considered is a multilayer perceptron (MLP) composed of two fully connected layers (784x400, 400x10) used for classification on the MNIST dataset with a recognition rate of 97.71% for the SNN model (96.63% for the equivalent CNN).

In terms of spiking benefits over ANN, FPGA implementation results on ZCU102 (Fig. 2) confirm the fact that SNN allows drastic DSP block reduction. It also uses less BRAM (for this

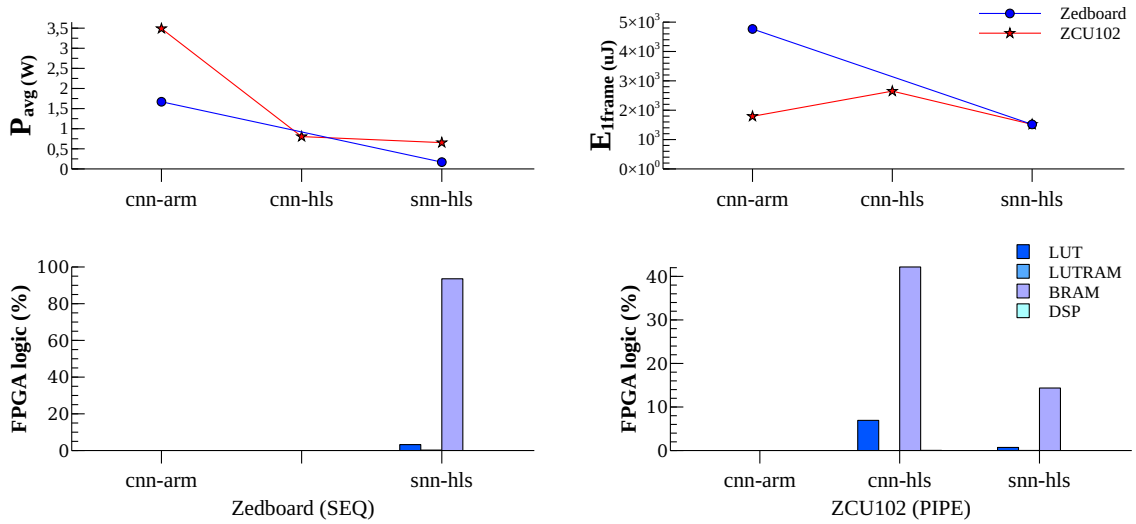

Fig. 2. FPGA logic occupation, average power and energy per frame for MLP benchmark

Table 1. Execution time (1 frame) for MLP sequential and parallel accelerators (100MHz)

MLP	ZedBoard (SEQ)			ZCU102 (PAR)		
T_{ex} (ms)	T_{min}	T_{avg}	T_{max}	T_{min}	T_{avg}	T_{max}
mlp_cnn_hls	–	–	–	3.29	**3.30**	7.67
mlp_snn_hls	2.27	**8.97**	77.83	0.59	**2.32**	20.12

Table 2. Execution time (1 frame) for MNIST sequential and parallel accelerators (100MHz)

MNIST	ZedBoard (SEQ)			ZCU102 (PAR)		
T_{ex} (ms)	T_{min}	T_{avg}	T_{max}	T_{min}	T_{avg}	T_{max}
mnist_cnn_hls	66.30	**66.32**	72.42	2.40	**2.41**	6.74
mnist_snn_hls	23.35	**81.98**	686.3	6.01	**21.10**	176.7
mnist_snn_vhdl	–	–	–	2.50	**5.70**	52.80

particular topology composed exclusively of fully connected layers) than the CNN which, indeed, does not fit in ZedBoard due to memory resources.

The actual measurement of inference times reveals that the SNN is somewhat faster than the ANN (respectively 2.3 ms and 3.3 ms per frame). However, there are very large variations in execution time for the SNN reaching almost a factor of 100 (Table 1). These variations are nearly negligible in comparison to MLP CNN showing much less fluctuation in execution time.

Concerning power, the benefits of FPGA logic reduction reflect well with an improvement of 19% (654 mW for SNN vs. 804 mW for CNN). In the end, the average energy cost is 53% less for the SNN (1517 uJ) compared to the equivalent CNN (2643 uJ) in the results of Table 5. Finally, if we compare these values to a pure software execution of the MLP CNN on ARM Cortex-A53 (ZCU102), the SNN consumes 15% less (1.5 mJ against 1.8 mJ). CNN ARM is rather effective in this particular case because the Cortex-A53 (1.2 GHz) is well suited to the processing of typical high-speed MAC operations found in fully connected layers.

B. MNIST

The second benchmark is a five layer CNN similar to LeNet5 applied to the MNIST dataset, composed of two convolutional layers (24x24x20, 8x8x40), two max-pooling layers (12x12x20, 4x4x40) and two fully connected layers (640x400, 400x10). The corresponding classification accuracy is 99.05% for SNN (97.92% for the equivalent CNN).

Comparing SNN against ANN implementation, SNN HLS outperforms its CNN equivalent with substantial savings of DSP blocks but also fewer LUTs. Like previously for MLP, the SNN consumes globally less FPGA logic but it does need a little more memory resources (BRAM, LUTRAM) for the proper management of membrane potential data (in particular additional storage is needed for complex spiking convolutions).

Unlike MLP, SNN HLS is significantly slower than the CNN (8.8 times) with respectively 21.1 ms and 2.4 ms per frame. As with the MLP model, inference time variations are very large for the SNN compared with that of the CNN, reaching almost a factor of 30. These problems originate from spike encoding and parallelism, and will be further addressed in section 4.5.

As for power, the SNN saves up to 42% over the CNN (respectively 669 mW and 1153 mW) but this does not result in actual energy savings due to high inference latency as mentioned previously. In the end, SNN HLS consumes 5.1 times more energy than CNN HLS (14 mJ vs. 2.7 mJ), and consumes even a bit more than CNN on ARM Cortex-A53 (14 mJ vs 13.7 mJ).

The topology used with the GTSRB dataset consists of three convolutional layers (28x28x6, 10x10x16, 1x1x120), two max-pooling layers (14x14x6, 10x10x16), and two fully connected layers (120x84, 84x43). Classification accuracy is 85.20% for

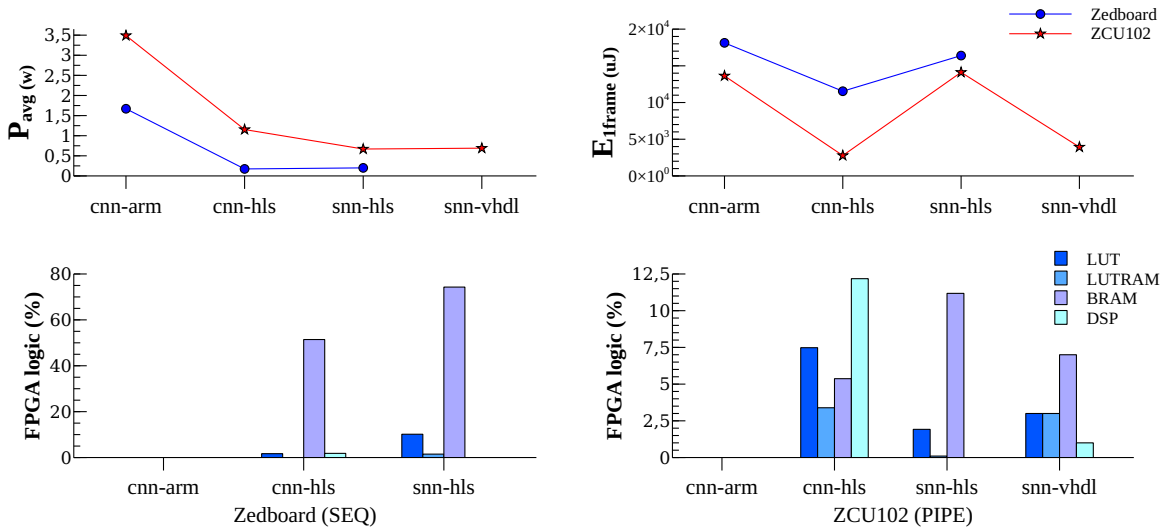

Fig. 3. FPGA logic occupation, average power and energy per frame for MNIST benchmark

this benchmark with the SNN model (91.99% for the equivalent CNN).

Regarding GTSRB, the actual implementation results of Fig. 4 report a significant decrease in FPGA logic for the SNN. This actually comes from the fact that i) the SNN achieves a relatively low level of parallelization and ii) the CNN is able to exploit much more parallelism than the SNN (Cf. usage of LUTs in Figure 4), especially for convolutions and max pooling. This, in turn, reflects logically in an execution time difference of nearly a factor of 100 advocating for the CNN (1.248 ms vs. 103.9 ms for SNN, Table 3).

C. GTSRB

A direct consequence is that, despite a significant decrease in logic resources and the associated power (2.113 W and 0,672 W respectively for CNN and SNN), energy cost ends up 26 times higher for the SNN (2.6 mJ vs. 70 mJ) due to execution times, and even exceeds greatly (8.9 times) the energy of the CNN on ARM Cortex-A53 (Table 5).

Table 3. Execution time (1 frame) for GTSRB sequential and parallel accelerators (100MHz)

GTSRB	ZedBoard (SEQ)			ZCU102 (PAR)		
T_{ex} (ms)	T_{min}	T_{avg}	T_{max}	T_{min}	T_{avg}	T_{max}
gtsrb_cnn_hls	24.17	**24.20**	30.24	1.25	**1.25**	6.07
gtsrb_snn_hls	31.86	**308.6**	2710	10.73	**103.9**	912.2

D. CIFAR-10

The last benchmark, which is also the biggest topology, consists of six convolutional layers (32x32x32, 32x32x32, 16x16x64, 16x16x64, 8x8x128, 8x8x128), three max-pooling layers (16x16x32, 8x8x64, 4x4x128) and one fully connected layer (2048x10). It is applied to the CIFAR-10 dataset with a classification accuracy of 79.2% for SNN (88.31% for the equivalent CNN). For this topology, both CNN and SNN designs exceed the resource capacity of a ZedBoard platform, so the following measurement results are reported for ZCU102 only.

The CIFAR-10 benchmark confirms again a global decrease of FPGA logic to the benefit of the SNN, greatly due to the complete absence of DSP blocks (Figure 5). This logically translates into a power reduction of 75% (742 mW for SNN and 3 W for CNN).

Table 4. Execution time (1 frame) for CIFAR-10 parallel accelerators (100MHz)

CIFAR-10	ZCU102 (PAR)		
T_{ex} (ms)	T_{min}	T_{avg}	T_{max}
cifar_cnn_hls	17.96	**17.96**	23.70
cifar_snn_hls	79.83	**462.8**	2315

However, as observed previously for the SNN, inference times are greatly impaired (463 ms against 18 ms for CNN, Table 4). This takes the preponderance over previous power savings and the SNN ends up consuming 6 times more in the final energy balance (343 mJ for SNN and 54 mJ for CNN), which remains however two times less than that of CNN on ARM Cortex-A53 (690 mJ, Table 5).

E. Efficiency analysis

In light of previous implementation results on FPGA, a first outcome is that SNNs may likely be less efficient than CNNs, or at best, they may approach the same efficiency level, even considering that the quality of HLS results is behind those of manual RTL. The only exception in our measurements is for MLP which is the smallest example composed exclusively of fully connected layers, but whenever the topology includes more

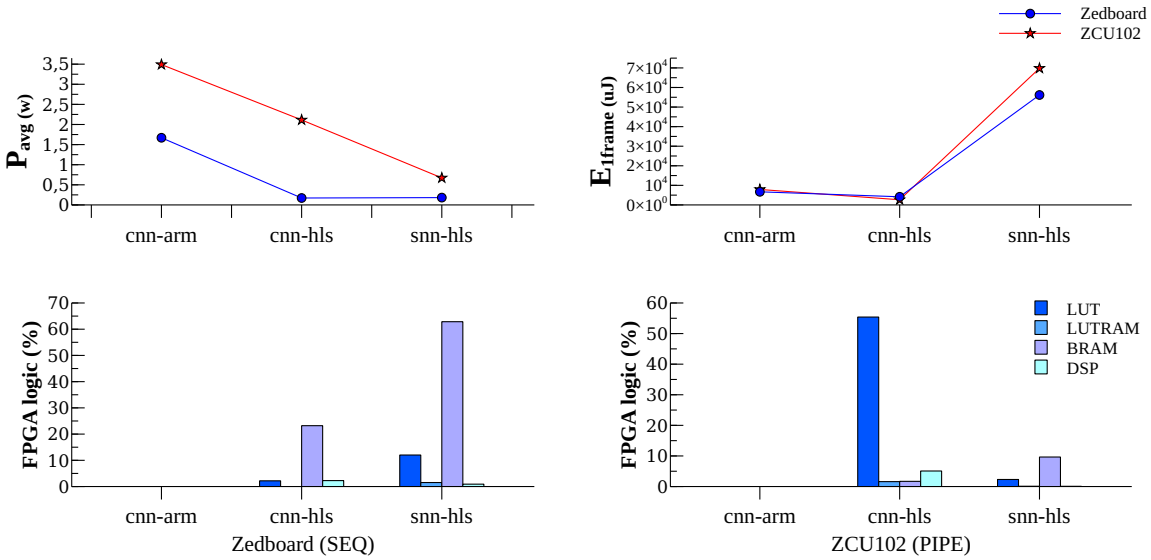

Fig. 4. FPGA logic occupation, average power and energy per frame for GTSRB benchmark

Table 5. Efficiency comparison of CNN and SNN implementations.

	ZedBoard			ZCU102		
	T_{avg} (ms)	P_{avg} (W)	E_{1frame} (mJ)	T_{avg} (ms)	P_{avg} (W)	E_{1frame} (mJ)
mlp_cnn_arm	2,85	1,67	4,76	0,51	3,49	1,78
mlp_cnn_hls	–	–	– [1]	3,29	0,80	2,65
mlp_snn_hls	8,97	0,17	1,52	2,32	0,65	**1,52**
mnist_cnn_arm	10,86	1,67	18,14	3,93	3,49	13,72
mnist_cnn_hls	66,32	0,17	11,54	2,41	1,15	**2,78**
mnist_snn_hls	81,98	0,20	16,40	21,10	0,67	14,12
mnist_snn_vhdl	–	–	–	5,70	0.69	3,93
gtsrb_cnn_arm	4,04	1,67	6,75	2,26	3,49	7,89
gtsrb_cnn_hls	24,20	0,17	4,16	1,25	2,11	**2,64**
gtsrb_snn_hls	308,6	0,18	56,17	103,9	0,67	69,82
cifar_cnn_arm	–	–	–	197,8	3,49	690,2
cifar_cnn_hls	–	–	–	17,96	3,00	**53,95**
cifar_snn_hls	–	–	–	462,8	0,74	343,4

[1] Empty entries mean that the networks are too large for the FPGA device.
[2] The average energy is obtained by multiplying T_{avg} with P_{avg}.

complex convolution layers the gap is growing to the benefit of CNNs and this is likely to increase with complex topologies (cf. Figure 5). The SNNs used in this study are greatly impaired by their execution times and this is for two main reasons: spike encoding and parallelism.

First, the coding scheme used, i.e. the inherent encoding of spike trains, is based on rate coding where the frequency of spike trains depends on the signal information (high signal intensity results in a high spike frequency, low-intensity results in low spike frequency). Rate (or frequency) coding infers relatively larger spike trains and higher activity, which reflects in execution times (and variations).

Another factor in the inference latency problem is related to the inner data processing and parallelism of SNNs. Effective spike processing requires numerous parallel and random accesses to large data memories, especially for the membrane potential of neurons. Parallelism is hindered by the fact that non-trivial memory partitioning and loop scheduling schemes are not always well identified by HLS tools. For instance, it works very well for MLP because fully connected layers are based on typical MAC operations very similar to digital filtering. However, it fails for convolutions, where accumulation operations occur irregularly at any location and any time depending on input firing spike events. HLS struggles to extract enough loop parallelism in these conditions and the problem quickly grows when scaling to bigger topologies. This is stressed when comparing HLS against manual implementations (MNIST) or sequential versus parallel performances for SNN HLS in Table 5.

And lastly, sparsity, a key feature of theoretical spike efficiency, turns out to be an issue for practical implementation in current silicon technology. Sparse, event-driven computations mean less and lighter activity which also comes out with large amounts of idling time. This is a problem for standard CMOS technology, especially for FPGAs. For instance, the level of static power is 623 mW for ZCU102 which represents around 80% - 90% of the total power budget on large SNNs like GTSRB or CIFAR-10. DNNs are generally big designs resulting in large amounts of static power resulting from leakage current in CMOS circuits and further amplified in FPGAs by the programmable interconnect network. With such levels of overconsumption, it is probable that the actual integration in silicon (and even more in FPGAs) limits greatly if not loses completely, the entire potential benefits of SNNs, and this problem is likely to be relevant even if a better spike encoding scheme than rate coding is used.

F. Comparison with other works

In a way to assess the relevance of SNN accelerators designed by HLS against existing works, we address in the following a comparison with other acceleration platforms. As most SNN related studies are evaluated on the MNIST dataset using MLP or smaller CNN networks such as LeNet, Table 6 compares with recent SNN FPGA platforms on close topologies, we include a few CNN GPU implementations as well.

First, the accuracy level is about 97% for MLP and 99% for

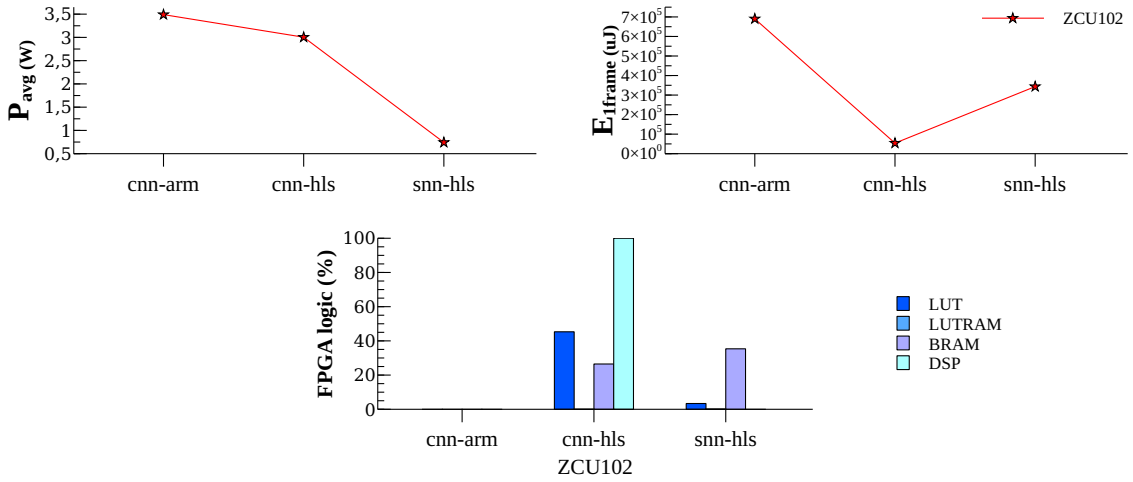

Fig. 5. FPGA logic occupation, average power and energy per frame for CIFAR-10 benchmark

LeNet like topologies in all implementations, enabling functional validation of the SNN accelerators developed using HLS.

In terms of performance, for MLP topologies the SNN HLS platform (line 2 of Table 6) and other implementations are at similar level (2.32ms vs. 6.21ms). Concerning spiking CNN topologies, the SNN HLS platform (last line of Table 6) is sensitively less effective than other FPGA implementations. It should be noted here that these accelerators run at higher frequencies (150 or 200 MHz). If we compare more specifically SNN HLS with [22], which is based on the same architectural model but designed manually, it is 3.7 times slower (at 100 MHz both). As stated in the efficiency analysis (section 4.5), HLS struggles to address efficiently the parallelism potential within spike processing layers and this reflects in the inference time. Aside from this, the level of performances is inline with those of state of the art implementations (including equivalent embedded CNN GPU acceleration, line 4 of Table 6).

5. CONCLUSION AND PERSPECTIVES

Other studies also started to appear recently pointing out an unanticipated inefficiency of SNN implementations against their equivalent ANN [26]. A first direction for improvements concerns alternative spike encoding methods in a way to ensure a reduced spiking activity, shorter inference times, and fewer variations. This can be based for instance on late research investigating other (temporal) spike coding schemes and associated learning methods ensuring lower firing rates [27]. Nevertheless, it also seems unavoidable to limit static and idle power consumption to really benefit from the high sparsity of computations in SNNs. Further work will check the efficiency of SNNs for ASIC implementation and also seek how to determine adapted power saving and management solutions (clock gating, power gating, DVFS, partial reconfiguration/blanking, ...) to address this sparsity / idle consumption issue. Concerning parallelism, future works will investigate the use of different synthesis tools and try to extend the level of loop parallelization efficiency when processing characteristic non-trivial accesses to large multidimensional arrays in spike processing. Since the actual state of this research on SNN efficiency is limited by these different questions, we will also continue to address ANNs and explore the use of partial reconfiguration techniques (partitioning, scheduling, DVFS, blanking, ...) to improve significantly their energy efficiency in FPGA implementations [28].

REFERENCES

1. The shift project, Implementing digital sufficiency, 2020, https://theshiftproject.org/
2. Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, Yoshua Bengio, Binarized Neural Networks: Training Neural Networks with Weights and Activations Constrained to +1 or -1, 2016.
3. Wolfgang Maass, Networks of spiking neurons: The third generation of neural network models, Neural Networks, 1997.
4. Powering the Edge: Driving Optimal Performance with the Ethos-N77 NPU, 2019, https://developer.arm.com/
5. Arm Ethos-U55 NPU, 2021, https://developer.arm.com/
6. Ankur Agrawal, Sae Kyu Lee, Joel Silberman, Matthew Ziegler, Mingu Kang, Swagath Venkataramani, Nianzheng Cao, et al., 9.1 A 7nm 4-Core AI Chip with 25.6TFLOPS Hybrid FP8 Training, 102.4TOPS INT4 Inference and Workload-Aware Throttling, IEEE International Solid- State Circuits Conference (ISSCC), 2021.
7. Eustace Painkras, Luis A. Plana, Jim Garside, et al., SpiNNaker: A 1-W 18-core system-on-chip for massively-parallel neural network simulation, IEEE Journal of Solid-State Circuits, 2013.
8. Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, et al., TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2015.
9. Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham

Table 6. Comparison of NN acceleration platforms on MNIST dataset

Work	*Platform*	*Frequency*	*Network*	*Topology*	T_{1frame} (ms)	*Power* (W)	E_{1frame} (mJ)	*Accuracy* (%)
[24] (2020)	**FPGA** Xilinx ZC706	200MHz	MLP SNN (Converted, 16 bits)	784-1024-1024-10	6,21	0,477	2,96	97,06
This work	**FPGA** Xilinx ZCU102	100MHz	MLP SNN (Converted, 8 bits)	784-400-10	2,32	0,654	1,52	97,7
[25] (2020)	**FPGA** Xilinx ZCU102	150MHz	Conv. SNN (Converted, 8 bits)	28x28-64C5-P2-64C5-P2-128-10	6,11	4,60	28,11	98,94
[23] (2020)	**GPU** Nvidia RTX 5000	1,62GHz	Conv. SNN (Spike Based, 16 bits)	28x28-32C3-P2-32C3-P2-256-10	1,15	61,2	70,38	99,2
	Edge GPU Nvidia AGX Xavier	1,37GHz			4,74	14	66,36	99,2
	FPGA Xilinx ZCU102	125MHz			0,47	4,51	2,12	99,2
[22](2020)	**FPGA** Xilinx ZCU102	100MHz	Conv. SNN (Spike Based, 8 bits)	28x28-20C5-P2-40C5-P2-400-10	5,70	0,690	3,93	99.05
This work	**FPGA** Xilinx ZCU102	100MHz	Conv. SNN (Converted, 8 bits)	28x28-20C5-P2-40C5-P2-400-10	21,10	0,669	14,12	99,05

Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al., Loihi: A Neuromorphic Manycore Processor with On-Chip Learning, IEEE Micro, 2018.

10. Murad Qasaimeh, Kristof Denolf, Jack Lo, Kees Vissers, Joseph Zambreno, Phillip H. Jones, Comparing Energy Efficiency of CPU, GPU and FPGA Implementations for Vision Kernels, IEEE International Conference on Embedded Software and Systems (ICESS), 2019.
11. Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, Vivienne Sze, Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks, IEEE Journal of Solid-State Circuits, 2017.
12. Ahmad Shawahna, Sadiq M. Sait, Aiman El-Maleh, FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review, IEEE Access, 2019.
13. Xilinx, DPU for Convolutional Neural Network, Concurrency and Computation Practice and Experience, 2019.
14. Andrew Boutros, Eriko Nurvitadhi, Rui Ma, Sergey Gribok, Zhipeng Zhao, James C. Hoe, Vaughn Betz, Martin Langhammer, Beyond Peak Performance: Comparing the Real Performance of AI-Optimized FPGAs and GPUs, International Conference on Field-Programmable Technology (ICFPT), 2020.
15. Stylianos I. Venieris, Christos-Savvas Bouganis, fpgaConvNet, 2017, http://cas.ee.ic.ac.uk/people/sv1310/fpgaConvNet.html
16. Francesco Conti, Daniele Palossi, Andrea Marongiu, Davide Rossi, Luca Benini, Enabling the heterogeneous accelerator model on ultra-low power microcontroller platforms, Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016.
17. Sebastian Höppner, Bernhard Vogginger, Yexin Yan, Andreas Dixius, Stefan Scholze, Johannes Partzsch, et al., Dynamic Power Management for Neuromorphic Many-Core Systems, IEEE Transactions on Circuits and Systems, 2019.
18. Fabian Schuiki, Michael Schaffner, Frank K. Gürkaynak, Luca Benini, A Scalable Near-Memory Architecture for Training Deep Neural Networks on Large In-Memory Datasets, IEEE Transactions on Computers, 2019.
19. IBM, The hardware behind analog AI, 2020, https://analog-ai-demo.mybluemix.net/hardware
20. IBM, IBM Unveils World's First 2 Nanometer Chip Technology, Opening a New Frontier for Semiconductors, 2008, https://newsroom.ibm.com/
21. iSoC, 2021, https://isoc-design.com/
22. Nassim Abderrahmane, Hardware design of spiking neural networks for energy efficient brain-inspired computing, Ph.D. dissertation, 2020.
23. Haowen Fang, Zaidao Mei, Amar Shrestha, Ziyi Zhao, Yilan Li, Qinru Qiu,Encoding, Model, and Architecture: Systematic Optimization for Spiking Neural Network in FPGAs, 39th International Conference on Computer-Aided Design 'ICCAD), 2020.

24. Jianhui Han, Zhaolin Li, Weimin Zheng, Youhui Zhang, Hardware implementation of spiking neural networks on FPGA, Tsinghua Science and Technology, 2020.
25. Xiping Ju, Biao Fang, Rui Yan, Xiaoliang Xu, Huajin Tang, An FPGA Implementation of Deep Spiking Neural Networks for Low-Power and Fast Classification, Neural Computation, 2020.
26. Xiping Ju, Biao Fang, Rui Yan, Xiaoliang Xu, Huajin Tang, Comparison of Artificial and Spiking Neural Networks on Digital Hardware, Frontiers in Neuroscience, 2021.
27. Saeed Reza Kheradpisheh, Timothée Masquelier, S4NN: Temporal Backpropagation for Spiking Neural Networks with One Spike per Neuron, International Journal of Neural Systems, 2020.
28. Robin Bonamy, Sebastien Bilavarn, Fabrice Muller, François Duhem, Simon Heywood, Philippe Millet, Fabrice Lemonnier, Energy Efficient Mapping on Manycore with Dynamic and Partial Reconfiguration: Application to a Smart Camera, International Journal of Circuit Theory and Applications, 2018, http://fortress-toolbox.unice.fr/