A Multi-Precision Bit-Serial Hardware Accelerator IP for Deep Learning Enabled Internet-of-Things

A Multi-Precision Bit-Serial Hardware Accelerator IP for Deep Learning Enabled Internet-of-Things

Maurizio Capra*, Francesco Conti†, Maurizio Martina*
*Department of Electronics and Telecommunication (DET) *Politecnico di Torino,* Turin, Italy, Email: name.surname@polito.it
†Energy-Efficient Embedded Systems Laboratory, *University of Bologna*, Bologna, Italy

***Abstract*—Deep Neural Networks (DNNs) computation-hungry algorithms demand hardware platforms capable of meeting rigid power and timing requirements. We introduce the Serial-MAC-engine (SMAC-engine), a fully-digital hardware accelerator for inference of quantized DNNs suitable for integration in a heterogeneous System-on-Chip (SoC). The accelerator is completely embedded in the form of a Hardware Processing Engine (HWPE) in the PULPissimo platform, a RISCV-based programmable architecture that targets the computational requirements of IoT applications. The SMAC-engine supports configurable precision for both weights (8/6/4 bits) and activations (8/4 bits), with scalable performance. Results in 65 nm technology demonstrate that the serial-MAC approach enables the accelerator to achieve a maximum throughput of 14.28 GMAC/s, consuming 0.58 pJ/MAC @ 1.0 V when operating at a precision of 4 bits for weights and 8 bits for activations.**

***Index Terms*—VLSI, hardware accelerator, Deep Learning, energy efficiency, bit-serial, DNN, CNN, low power**

I. Introduction

Today, Artificial Intelligence (AI) is increasingly permeating many aspects of our society, gaining attention and unleashing powerful applications that go beyond all expectations from just a few years ago. Deep Learning (DL) enables a large variety of ad-hoc brain-inspired applications such as image and speech recognition [1] [2], object detection and segmentation [3] or financial forecast [4]. Even if Deep Learning has a high computational cost with respect to conventional Machine Learning (ML), it has strongly emerged in recent years. The reason why DL is taking off compared to traditional ML lies in its capability to process and take advantage of a huge amount of data. Despite the data availability increase, ML has no significant performance-enhancing, while the counterpart can exploit its deep layer design to boost the task outcome performance. Moreover, the continuous digitization concerning the reality surrounding us produces an outstanding amount of information from which the DL can draw and thrive tirelessly. However, such remarkable accomplishment comes at a cost represented by the fact that constant data stream combined with challenging algorithms make DL applications computation-hungry. This problematic landscape poses an obstacle for the Internet-of-Things (IoT) future development since many of its constituent smart nodes have limited resources. Edge computing intervenes by cutting off the need for data transmission and broad bandwidth by pushing the analytic part from servers to sensors and portable devices, encouraging the integration of accelerators next to the data sources.

In this scenario, the call for suitable architectures able to tackle both computation and power demands arises. Hardware accelerators can play an essential role in enabling the deployment of intense inference sessions on Application Specific Integrated Circuit (ASIC)-scale devices [5]. Software and hardware co-design techniques are heavily employed to cut off Deep Neural Networks (DNNs) complexity, along with the energy required per operation such as quantization [6]. In addition, architectural strategies exploit the data spatial locality of the DNN to reuse activations or weights. Consequently, this reduces the required memory bandwidth and leads to stationary Datapaths (DPs).

In this work, we present the Serial-MAC-engine (SMAC-engine), a 65 nm hardware accelerator with wide flexible parallelism that targets quantized DNNs: 8 or 4 bits for activations and 8, 6, or 4 bits for weights (considered the standard in the inference at the edge [7]). The main contributions of this paper are:

- A parametric DP easy to be adapted to any dataflow. In this case, an input/output stationary DP in which activations are shared and reused, dedicating the whole available memory bandwidth to the weights loading process.
- The serial attitude allows scaling between the various parallelisms, reaching almost 100% of the hardware utilization, w.r.t. parallel solutions. The consumption of 0.58 pJ/MAC only makes the power budget of the order of mW, suitable for IoT.
- The proposed solution is equipped with a memory-based interface and integrated into a heterogeneous System-on-Chip (SoC), like PULPissimo [8], in which one or many cores can orchestrate the processing of the data directly coming from the sensors.
- This work is entirely open-source available at [9].

In Section II, we introduce the most promising work found in the literature. Section III describes the architectural features and the design choices from the basic block up to the final integrated engine. In Section IV, performance concerning frequency and synthesis are outlined in detail (see Table I) as well as the validation performed on popular DNN topologies (VGG16 [10] and SqueezeNet [11]). Finally, Section V sums up the work and suggests some future developments.

II. Related work

During the last years, many efforts have been lavished to produce cutting-edge hardware. Computation-hungry DL applications combined with IoT expansion have started the race for the edge computing development. This last offers a tradeoff among several factors, such as performance, accuracy, and power consumption. The main bottleneck for energy is the continuous stream of data coming from the memory as an effect of DNN algorithms greed for activations and weights. Based on this assumption, two main possible solutions can be adopted (often used together): to reduce either the amount (pruning [20]) or bitwidth (quantization [21] [22]) of data and to reduce the data movements across the architecture [5]. Quantization allows cutting down the bit-precision of activations and weights, reducing the memory and computational complexity enormously. Concerning such techniques, several software approaches tried to provide efficient libraries to boost quantized DNN process over microcontroller unit architectures such as CMSIS-NN [12] and PULP-NN [13]. Despite both of them being oriented towards maximizing the overall performance but minimizing the memory footprint, CMSIS-NN targets the Arm Cortex-M processors, while PULP-NN (based on the previous one) targets the PULP SoCs [23] [8] [24].

To better exploit quantization, new bit-serial architectures have been proposed like UNPU [17], Stripes [18], and Loom [19]. These

TABLE I
COMPARATIVE TABLE.

HW solution		Technology [nm]	Frequency [MHz]	Area [mm^2]	Throughput [GMAC/s]	Power (@ 25°C) [mW]	Energy Efficiency [pJ/MAC]	Throughput/Area [GMAC/s/mm^2]
SW	CMSIS-NN [12]	-	-	-	0.05 - 0.15	1 - 1000	667 - 2000	-
	PULP-NN [13]	-	-	-	0.5 - 1	1 - 100	40 - 67	-
bit-parallel	ShiDianNao [14]	65	1000	4.86	64	320	5	13.17
	Eyeriss [15] @ 1 V	65	200	12.25	23	278	12.09	1.88
	BRein [16] @ 1 V	65	400	3.9	518	600	2	132.82
bit-serial	UNPU [17] @ 1.1 V	65	200	16	827	5.92	0.36	51.69
	Stripes [18]	-	-	16	345.6	297	0.86	21.6
	Loom [19]	-	-	-	-	-	-	-
	SMAC-E @ 1.0 V *	**65**	**476**	**0.36**	**14.28**	**7.38**	**0.58**	**40.8**
	SMAC-E @ 1.2 V *	**65**	**625**	**0.36**	**18.75**	**12.91**	**0.76**	**52.08**

* SMAC-E results refer to Pa=8 and Pw=4

allow to dynamically adapt to the incoming data bitwidth by enhancing hardware utilization compared to parallel solutions. Moreover, the serial approach requires straightforward arithmetic logic hardware that grants reduced area and power budgets, enabling the inference at the edge required by IoT.

A way to drastically reduce the data movement is to have a memory closely-coupled to the accelerator, avoiding deep hierarchy. UNPU [17] rely on a Tightly Coupled Data Memory (TCDM) through which it can be programmed and reach activations and weights. Data movement can also be minimized by adopting aggressive strategies of activations or weights reuse at the local level through the sub-components of the design, like in the case of Eyeriss [15] where just 0.0029 DRAM access/multiply occur. Exploiting a different paradigm called logic-in memory, it is possible to drastically reduce the memory accesses since the computation is performed directly inside the memory like in BRein [16].

III. ARCHITECTURE

A. Background

The fundamental operation for both fully connected and convolutional layers is the multiply-and-accumulate (MAC). Since tens of thousands of MACs per layer occur, the need for high performance requires to develop a highly parallel structure able to allocate millions or billions of operations per second. Commonly, the most used paradigms are two: temporal and spatial architectures. Temporal ones are typically represented by CPUs and GPUs, where clusters of Arithmetic-Logic Units (ALUs) are driven from a unique core control. ALUs are decoupled, making communication or data transfer impossible, indeed data only come from the memory hierarchy. On the other hand, hardware accelerators exploit spatial architectures where ALUs, also named Processing Elements (PE), are distributed, forming a network in which data can travel through local memory buffers. Since data fetch and relocation are two of the most expensive tasks in terms of energy, architecture designs are tailored to ad-hoc dataflows [5], which data reuse cuts off memory and energy consumption. Each MAC needs to fetch an activation and a weight, for a total of two memory accesses, another memory access is needed to write back the result. In order to overcome such bottleneck, input and output stationary approaches have been developed. The former consists in reading the activations or weights once and then retain these values by means of registers as much as possible. On the contrary, the output stationary approach minimizes the energy demanded to write and read partial sums to and from memory. In fact, by allocating accumulators, partial results are retained and reused locally.

In this work, we present an architecture that is both input and output stationary. Activations are shared among PEs and reused, while weights are continuously fetched. Three different accumulation levels avoid stressing the memory from the partial results stream as detailed in Subsection III-B.

Despite accelerators are specialized units able to perform just a few tasks, although, with high performance, they have to work in complex systems, such as a microprocessor. As a consequence, each accelerator must have a proper interface. In this current work, we integrated our SMAC-engine into the PULPissimo platform as a Hardware Processing Engine (HWPE) [25]. PULPissmo is a parallel ultra-low-power microcontroller architecture suitable to be embedded in an IoT endpoint [8]. Thus, the SMAC-engine would serve as a power-efficient extra-core for DNN edge processing. Its interface is composed of a TCDM and a microprogrammed Control Unit (CU). While the first is used by the central core to transfer data to the accelerator, the latter orchestrates the elaboration phase. The memory bandwidth towards the TCDM has been identified in 128 bits/cycle since this value ensures a good trade-off between the area and frequency of the engine as it will be further discussed in Subsection III-D.

With regard to the parallelism, the architecture aims to present a flexible solution able to process the state-of-the-art quantized neural networks found in the literature. Nowadays, thanks to quantization techniques, the required bitwidths have been identified in 8 and 4 bits for the activation width, called Pa, and in 8, 6, and 4 bits for the weights width, called Pw. The data bitwidth can be suitably modified (Pa and PW) in order to accommodate any application. The parallelism chosen in this work is now considered the state of the art for video applications and beyond [7]. Such choice allows not to deteriorate classification tasks of the DNNs while simultaneously speeding up the computation and strongly reducing the memory occupation, especially for the weights, which overall number tends to be greater than the number of activations.

For what concerns the data flow, we derived output and input stationary data flow in which convolution is performed by fetching the activations along the channels' direction. Algorithm 1 describes the data flow loops required to complete a convolution task using the typical sliding window technique, where x is the input feature map, y the output one, and w the kernel. Six nested loops are needed to cycle first over the channel depth, then over the kernel size, up to the feature maps size, moreover, two further loops serve for bit-

wise operations. Shifting first the activations and then the weights from the Least Significant Bit (LSB) towards the Most Significant Bit (MSB), the engine produces the partial results through bit-serial multiplications and sums. $n_W^{[i]}$, $n_H^{[i]}$, $n_C^{[i]}$ identify the width, height, and depth of the i-th feature map, while $f^{[i]}$ is the size of the i-th filter. Figure 1 is a representation of the above-mentioned data flow where activations and weights are fetched along the channel direction to produce output feature maps in the same orientation. Such an approach allows us to retain activations for as many kernels as possible inside the considered convolutional volume and locally accumulate partials sums until the end of the task.

Algorithm 1 Convolution nested loops and bit-serial loops.

$ConvolutionLoops:$

for $h_{out} = 0 : n_H^{[i]}$ **do**
  for $w_{out} = 0 : n_W^{[i]}$ **do**
    for $k_{out} = 0 : n_C^{[i]}$ **do**
      for $l = 0 : f^{[i]}$ **do**
        for $j = 0 : f^{[i]}$ **do**
          for $k_{in} = 0 : n_C^{[i-1]}$ **do**

$Bit - SerialLoops:$

for $wbit = LSB : MSB$ **do**
  for $xbit = LSB : MSB$ **do**

$$y[k_{out}][w_{out}][h_{out}] += x[k_{in}][w_{out}+l][h_{out}+j][xbit] \times W[k_{out}][k_{in}][l][j][wbit]$$

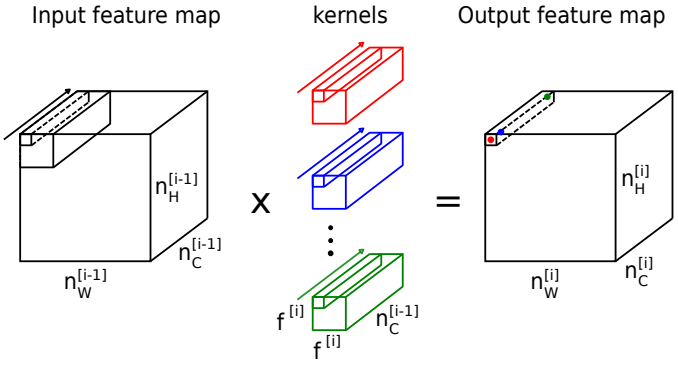

Fig. 1. Input/Output stationary data flow visualization.

B. SMAC

The SMAC's basic structure, Figure 2, is based on the one proposed by Sharify et al. named Loom [19] and by Lee et al. UNPU [17].

While we derived the main architecture from the first, we were inspired by the second in terms of energy efficiency and data bitwidth, however, using limited resources w.r.t. UNPU to make the architecture more suitable to the target platform (Subsection III-D).

Loom architecture was inspired in turn by DaDianNao [26], where the structure can perform M MAC operations per cycle concurrently. Nonetheless, allocating M multipliers and an adder tree would severely impact the final clock frequency, especially when M is large. By replacing the parallel structure with a serial one, multipliers can be substituted by AND gates. The throughput is kept unchanged by increasing the number of computational engines. With such a structure, M MAC operations, equivalent to the ones performed by the parallel structure, can be performed only thanks to two accumulators inserted after the AND block to retain the partial sums. Indeed, while the weight bits are stationary, the activation bits are shifted serially, and the partial sums relative to the fixed weight bit have to be summed together by a first accumulator AC1. The coherence among the partial sums is kept by performing a shift to the right. This is correct provided that activations and weights are shifted from the least significant bit (LSB) to the most significant bit (MSB), otherwise, the shifting would be performed in the opposite direction.

Following the same logic, a second accumulator AC2 has been inserted after the previous one to take care of the partial results obtained for each of the weight bit. Moreover, since data are represented in two's complement, a register between AC1 e AC2 allows inverting the content of AC1 whenever the weight bits are fixed to the MSB. Unlike AC1, AC2 is composed of 4 registers, this allows to work with four different filters in parallel and retain their partial sums. A third and final accumulator AC3 has been inserted to sum together partial results belonging to the same convolutional volume.

Ideally, after the convolution operation has been completed, the outcome must be quantized to be again represented with the same bitwidth of the input activations, thus with Pa bits. The amount of required shifting depends on the size of the specific convolutional layer. However, the idea of instantiating barrel shifters to perform quantization is not particularly convenient, mostly due to the area overhead requirements, which would lead to an unacceptable area occupation. A simple, but effective solution is to perform quantization serially by using shift registers in AC3 stage. Hence, once the convolution operation is done, a pre-loaded programmable counter can be exploited to serially shift the registers' values. Finally, after the quantization step, one of the most used activations function, the Rectified Linear Unit (ReLU), has been inserted. The ReLU is performed thanks to a multiplexer that uses as input selection the MSB of the convolution output.

C. SMAC-engine

The hardware datapath's width has been designed on the worst case, i.e., considering the maximum bitwidth for both activations and weights, Pa = 8 and Pw = 8.

Unlike the parallel solution, the serial one requires increased latency to perform M MACs. Instead of M MAC/cycle, it can perform M MACs after Pa×Pw cycles. Thus, for the serial solution in order to match the same throughput as the parallel one, the number of MAC blocks must be increased by Pa×Pw times. Since the serial solution employs far less hardware than the parallel counterpart, even instantiating more multipliers, the area remains reasonable.

To establish an acceptable value for M, a comparison in terms of area vs. operating frequency between parallel and serial approaches has been performed. Using the UMC-65 nm library in worst-case conditions, namely 0.9 V supply voltage @ 125 °C, we found that even if the SMAC block is replicated Pa×Pw times, the area for the parallel case is still 2.9× larger. Indeed, a SMAC block requires an order of magnitude less area than the parallel counterpart and is capable of reaching a maximum frequency that is nearly twice. This analysis has shown that a good trade-off among area, bandwidth, and frequency is obtained with M = 16, as described in Subsection III-D.

In conclusion, the SMAC-engine is a cluster of Pa×Pw = 64 SMAC blocks designed to support the worst-case (Pa = 8 and Pw = 8) and with M = 16. Activations are shared among SMACs, so the engine can process M activations per convolution, while each SMAC can take care of M weights per filter. Thus, the SMAC-engine can process 64 filters per convolution, that can be expanded to 256 thanks to the four registers in AC2 and AC3 stages. When more than 256 filters are involved, such as 512 for example, the structure handles splitting the operation into two parts.

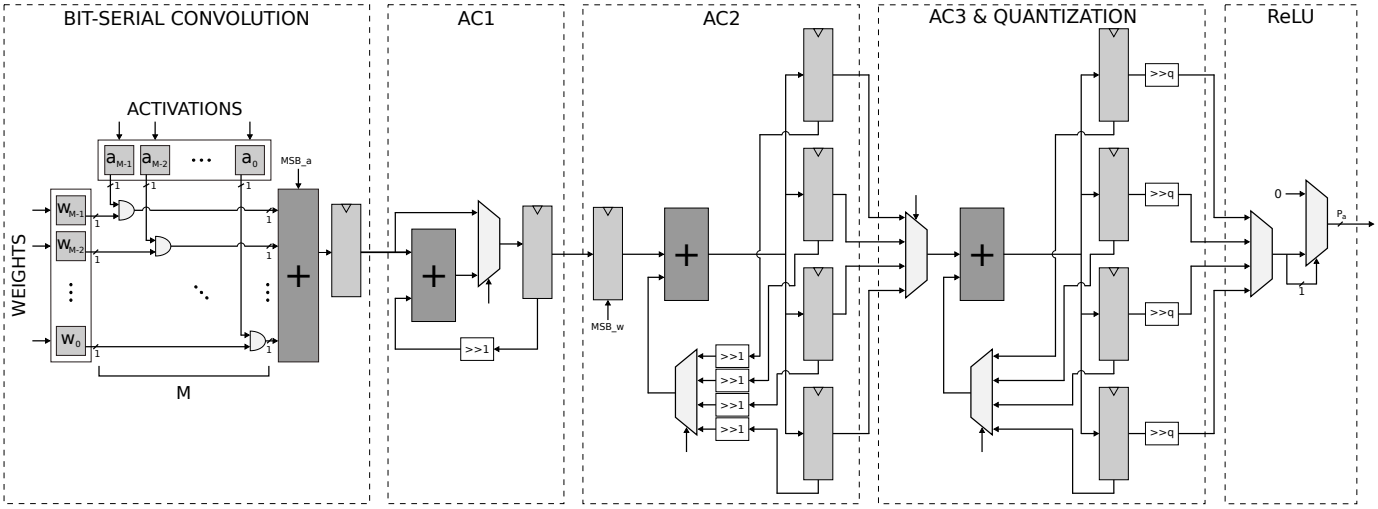

Fig. 2. Datapath architecture of a single SMAC block.

Besides the datapath, the SMAC-engine is equipped with a low-level Control Unit (CU) in charge of generating the control signals to orchestrate the convolution operation. It can work with kernels up to 3×3, but potentially it can be expanded to work with larger filters, provided that the accumulator AC3 is adequately sized to store more partial results. Low-level CU is composed of two main modules: a 16 states Finite State Machine (FSM) able to drive the DP progress, and a cluster of counters required to schedule the FSM evolution. The FSM generates status signals required by a high-level FSM that handles address generation and the memory interface as described in Subsection III-D.

D. PULPissimo-integration

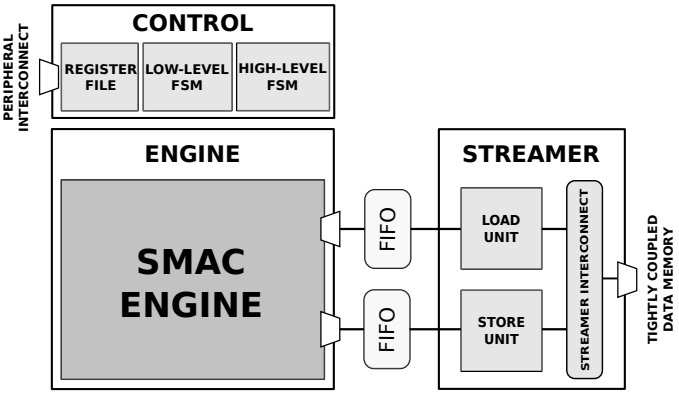

Fig. 3. HWPE architecture. The SMAC-engine is integrate in PULPissimo by means of a control and a Streamer. The first orchestrates the elaboration thanks to a Low-Level and a High-Level FSM and a register file, while the second manages the data streams.

One of this paper's main contributions is to provide a hardware accelerator fully integrated into a microcontroller-based platform capable of enabling DL in an IoT node. The target platform is PULPissimo since it offers the opportunity to integrate accelerators in the form of HWPE, and it is also entirely open-source. There is no direct communication between the processor present in PULPissimo and the HWPE, but data (activations and weights) are written in the TCDM by the former and fetched by the latter, and vice-versa. Since the TCDM is shared with other components present in the SoC, an interface mechanism that manages the accesses is needed, as described in [23]. Such an entity is called Streamer. It follows a handshake protocol to establish a directional connection through which data streams can flow. Whenever different clock domains are present, a FIFO can be used to connect a Load Unit and a Store Unit avoiding the leak of information. Besides the Streamer, another interface is necessary to program the accelerator by setting the counters' values accordingly to the size of the convolutional layer to be processed. This is why the HWPE presents a peripheral channel connected with a memory-mapped register file, as illustrated in Figure 3. The central core present in PULPissimo exploits such peripheral to upload setting values, which will then be used to program the counters (present in the low-level CU) and the microprogrammed high-level CU. At the beginning of the convolution operation, while activations and weights are uploaded to the engine, counters are updated with the corresponding value of the register file that depends on the current convolutional layer's size.

As previously stated, the maximum bandwidth is 128 bit/cycle. This bandwidth is related to how the Streamer handles the TCDM both during reading and writing operations. Considering the maximum activations and weights bitwidth, we can state that it is possible to fetch either 128/Pa = 16 activations or weights in a single cycle. This is another reason to choose M = 16.

Since the dataflow has been chosen to be input stationary, the SMAC-engine is built in a direction that maximizes the number of usable filters with the same activations. Explicitly, allocating 64 SMAC blocks sharing the same activations is feasible thanks to the serial approach in executing multiplications. Exploiting the available bandwidth to fetch 128 weight bits per cycle (instead of 128/Pw = 16 weights on the entire bitwidth), in Pa = 8 cycles (required to shift the activation bits) it is possible to achieve a total of 128×8 = 1024 bits, that is precisely the number of bits necessary to feed 64 SMAC blocks, each working with M = 16.

All the source files of the SMAC-engine and the instructions for its correct integration in Pulpissimo are available online [9].

IV. Results

The Serial-MAC-engine has been synthesized with a 65 nm UMC technology in worst-case conditions (0.9 V @125 °C) leading to 0.36 mm^2 silicon area coverage. To validate the architecture and define its behavior, VGG16 and SqueezeNet have been used as a test-bench implementing different parallelism widths. Table I compares the SMAC-engine with other state-of-the-art architectures devoted to DNN algorithms processing described in Section II. Considering that in convolutional networks, the basic operation is represented by a MAC, we used the energy efficiency metric defined as the energy needed to perform a complete MAC. The SMAC-engine with its 0.58 pJ/MAC can be compared with the efficiency expressed by UNPU [17]. Even if the latter has a much higher throughput, our solution features a 44.4× smaller area, fundamental in area-constrained platforms such as IoT edge nodes. Moreover, with the same silicon coverage, the performance would be almost identical, as depicted by the throughput to area ratio in the last column, Table I. Furthermore, while UNPU is a stand-alone accelerator, our work is already integrated in a microcontroller-based system and ready to use. The SMAC-engine outperforms Eyeriss [15] and BRein [16] in terms of energy efficiency, while being comparable to Stripes [18] that follows the same serial approach. Although the architecture was inspired in part by Loom [19], unfortunately, the latter does not offer an implementation on any technological node, making the comparison incomplete. The SMAC-engine, therefore, represents a possible detailed implementation of a Loom alike architecture.

Software methods like [12] and [13] offer high-level flexibility suitable for several different hardware platforms, boosting their applications. However, our ASIC-based approach represents a DNN-tailored hardware accelerator capable of optimizing every step from data transaction to the MAC execution.

The current design represents an example of how the SMAC-engine could scale. We fixed M = 16 and the maximum width of activations and weights to 8 bits, but potentially the scheme can be enlarged to accommodate more general cases as well as tailored on a specific instance where the optimization has to be pushed to the limit. Even the maximum kernel size can be modified by adjusting the AC3 width. In all the above-mentioned scaling paradigms, the DP bitwidth needs to be tuned according to the application to be tackled, while the CU remains unchanged since it can be programmed through the external peripheral. Additionally, as SMAC blocks are replicated to share activations, they can be replicated to share weights in a systolic array fashion (this will be investigated as future work).

Figure 4 illustrates how in VGG16 and SqueezeNet, the throughput, intended as the number of MAC per clock cycle, remains unchanged over the kernel sizes according to different values of Pa and Pw. Consequently, the bottleneck is represented by the data parallelism regardless of which layer is being processed. Therefore, knowing that, as mentioned above, the SMAC-engine can easily be scaled in all its parameters, the designer can adapt those features in such a way that the required throughput is met. The efficiency drop in Squeezenet's conv1 is due to the fact that the 7x7 kernel has been decomposed into two 3x3 kernels that slow down the execution.

Table II shows how data bitwidth affects the latency and the power consumption having as target the third layer of VGG16. This condition stresses the SMAC-engine to work with a number of filters greater than the default one (64), resulting in one of the most power-hungry configurations. The number of cycles refers to the clock latency required to load activations and weights, process the entire convolutional volume, and store the results back into the memory.

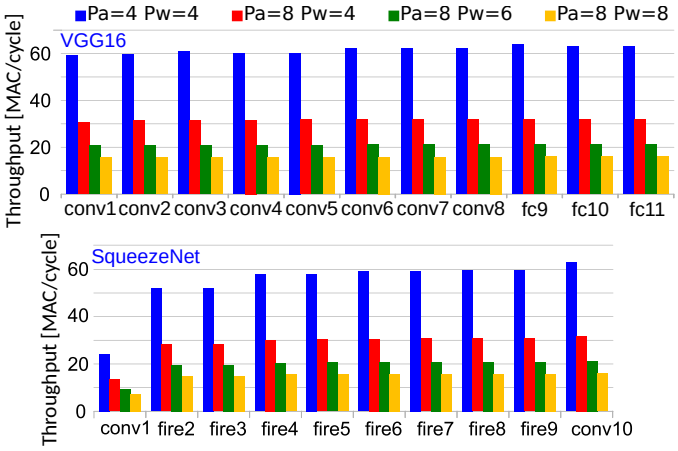

Fig. 4. Layers throughput over several bitwidth combinations.

TABLE II
Number of cycle and power consumption with different data bitwidth related to a filter of size $3 \times 3 \times 128 \times 128$.

Bitwidth	# cycle [M]	Throughput [MAC/cycle]		Power [mW]
		SMAC	SMAC-engine	
Pa=4, Pw=4	30.84	0.937	59.97	7.38
Pa=8, Pw=4	58.92	0.490	31.39	7.38
Pa=8, Pw=6	88.67	0.326	20.86	7.41
Pa=8, Pw=8	117.57	0.246	15.73	7.42

Results obtained @ 1.0 V with an operating frequency of 476 MHz

Since multiplications are performed serially, the throughput tends to decrease by increasing data width as depicted in the third and fourth columns, where both single SMAC and SMAC-engine (64 SMACs) are results reported. Even though the required clock cycles increase 4x going from the narrower bitwidth to the larger one, the power undergoes a negligible increase of just 40 µW, thanks again to the bit by bit multiplication. This is an excellent indicator that hardware usage is constant and there are no parts that waste energy when halving bitwidth.

V. Conclusion

This work introduces the Serial-MAC-engine (SMAC-engine), a 65 nm convolutional hardware accelerator with variable parallelism suitable for quantized neural networks. The 14.28 GMAC /s at the cost of 0.58 pJ/MAC @ 1.0 V have been achieved exploiting a serial approach where multiplication is performed bit by bit, reducing enormously the hardware required for multipliers and consequently boosting the maximum accessible frequency. The wide choice in the data parallelism enables the architecture to make inference on several applications and networks targeting the Internet-of-Things paradigm.

The accelerator is fully integrated into a microcontroller-based platform, namely PULPissimo. In such an SoC platform, the central core drives the SMAC-engine through its memory-based programmable interface specifying which kind of operation the accelerator must execute.

The source code and the instructions to integrate the accelerator in PULPIssimo are available online [9].

Acknowledgment

The authors would like to thank Mr. Mattia Carlo Petruzzellis for his precious work during his master thesis period. His attitude and expertise have been fundamental in order for the project to succeed.

REFERENCES

[1] E. Ichikawa, K. Sawada, K. Hashimoto, Y. Nankaku, and K. Tokuda, "Image recognition based on separable lattice hmms using a deep neural network for output probability distributions," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3021–3025, April 2018.

[2] J. P. Dominguez-Morales, Q. Liu, R. James, D. Gutierrez-Galan, A. Jimenez-Fernandez, S. Davidson, and S. Furber, "Deep spiking neural network model for time-variant signals classification: a real-time speech recognition approach," in *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, July 2018.

[3] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788, 2015.

[4] A. Maratkhan, I. Ilyassov, M. Aitzhanov, M. F. Demirci, and M. Ozbayoglu, "Financial forecasting using deep learning with an optimized trading strategy," in *2019 IEEE Congress on Evolutionary Computation (CEC)*, pp. 838–844, June 2019.

[5] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *CoRR*, vol. abs/1703.09039, 2017.

[6] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1," *CoRR*, vol. abs/1602.02830, 2016.

[7] H. Gao, W. Tao, D. Wen, T. Chen, K. Osa, and M. Kato, "Ifq-net: Integrated fixed-point quantization networks for embedded vision," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 720–7208, 2018.

[8] D. Rossi, F. Conti, A. Marongiu, A. Pullini, I. Loi, M. Gautschi, G. Tagliavini, A. Capotondi, P. Flatresse, and L. Benini, "Pulp: A parallel ultra low power platform for next generation iot applications," in *2015 IEEE Hot Chips 27 Symposium (HCS)*, pp. 1–39, Aug 2015.

[9] M. Capra, "Smac-engine." https://github.com/MaurizioCapra/SMAC-engine.git, 2020.

[10] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[11] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size," 2016.

[12] L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs," *arXiv e-prints*, p. arXiv:1801.06601, Jan 2018.

[13] A. Garofalo, M. Rusci, F. Conti, D. Rossi, and L. Benini, "PULP-NN: Accelerating Quantized Neural Networks on Parallel Ultra-Low-Power RISC-V Processors," *arXiv e-prints*, p. arXiv:1908.11263, Aug 2019.

[14] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pp. 92–104, June 2015.

[15] Y. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, pp. 127–138, Jan 2017.

[16] K. Ando, K. Ueyoshi, K. Orimo, H. Yonekawa, S. Sato, H. Nakahara, S. Takamaeda-Yamazaki, M. Ikebe, T. Asai, T. Kuroda, and M. Motomura, "Brein memory: A single-chip binary/ternary reconfigurable in-memory deep neural network accelerator achieving 1.4 tops at 0.6 w," *IEEE Journal of Solid-State Circuits*, vol. 53, pp. 983–994, April 2018.

[17] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H. Yoo, "Unpu: An energy-efficient deep neural network accelerator with fully variable weight bit precision," *IEEE Journal of Solid-State Circuits*, vol. 54, pp. 173–185, Jan 2019.

[18] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–12, 2016.

[19] S. Sharify, A. D. Lascorz, P. Judd, and A. Moshovos, "Loom: Exploiting weight and activation precisions to accelerate convolutional neural networks," *CoRR*, vol. abs/1706.07853, 2017.

[20] K. Nan, S. Liu, J. Du, and H. Liu, "Deep model compression for mobile platforms: A survey," *Tsinghua Science and Technology*, vol. 24, pp. 677–693, Dec 2019.

[21] J. Choi, Z. Wang, S. Venkataramani, P. I-Jen Chuang, V. Srinivasan, and K. Gopalakrishnan, "PACT: Parameterized Clipping Activation for Quantized Neural Networks," *arXiv e-prints*, p. arXiv:1805.06085, May 2018.

[22] D. Zhang, J. Yang, D. Ye, and G. Hua, "LQ-Nets: Learned Quantization for Highly Accurate and Compact Deep Neural Networks," *arXiv e-prints*, p. arXiv:1807.10029, Jul 2018.

[23] F. Conti, R. Schilling, P. D. Schiavone, A. Pullini, D. Rossi, F. K. Gürkaynak, M. Muehlberghuber, M. Gautschi, I. Loi, G. Haugou, S. Mangard, and L. Benini, "An iot endpoint system-on-chip for secure and energy-efficient near-sensor analytics," *CoRR*, vol. abs/1612.05974, 2016.

[24] E. Flamand, D. Rossi, F. Conti, I. Loi, A. Pullini, F. Rotenberg, and L. Benini, "Gap-8: A risc-v soc for ai at the edge of the iot," in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 1–4, July 2018.

[25] F. Conti, P. D. Schiavone, and L. Benini, "Xnor neural engine: A hardware accelerator ip for 21.6-fj/op binary neural network inference," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[26] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "Dadiannao: A machine-learning supercomputer," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 609–622, Dec 2014.