XNOR Neural Engine: a Hardware Accelerator IP for 21.6 fJ/op Binary Neural Network Inference

Author(s):
Conti, Francesco; Schiavone, Pasquale D.; Benini, Luca

Publication date:
2018-11

Permanent link:
https://doi.org/10.3929/ethz-b-000279119

Originally published in:
IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 37(11), https://doi.org/10.1109/TCAD.2018.2857019

XNOR Neural Engine: a Hardware Accelerator IP for 21.6 fJ/op Binary Neural Network Inference

Francesco Conti, *Member, IEEE,* Pasquale Davide Schiavone, *Student Member, IEEE,* and Luca Benini, *Fellow, IEEE*

***Abstract*—Binary Neural Networks (BNNs) are promising to deliver accuracy comparable to conventional deep neural networks at a fraction of the cost in terms of memory and energy. In this paper, we introduce the XNOR Neural Engine (XNE), a fully digital configurable hardware accelerator IP for BNNs, integrated within a microcontroller unit (MCU) equipped with an autonomous I/O subsystem and hybrid SRAM / standard cell memory. The XNE is able to fully compute convolutional and dense layers in autonomy or in cooperation with the core in the MCU to realize more complex behaviors. We show post-synthesis results in 65nm and 22nm technology for the XNE IP and post-layout results in 22nm for the full MCU indicating that this system can drop the energy cost per binary operation to 21.6fJ per operation at 0.4V, and at the same time is flexible and performant enough to execute state-of-the-art BNN topologies such as ResNet-34 in less than 2.2mJ per frame at 8.9 fps.**

***Index Terms*—Binary Neural Networks, Hardware Accelerator, Microcontroller System**

I. Introduction

TODAY, *deep learning* enables specialized cognition-inspired inference from collected data for a variety of different tasks such as computer vision [1], voice recognition [2], big data analytics [3], financial forecasts [4]. However, this technology could unleash an even higher impact on ordinary people's life if it was not limited by the constraints of data center computing, such as high latency and dependency on radio communications, with its privacy and dependability issues and hidden memory costs. Low-power, embedded deep learning could potentially enable vastly more intelligent implantable biomedical devices [5], completely autonomous nano-vehicles [6] for surveillance and search&rescue, cheap controllers that can be "forgotten" in environments such as buildings [7], roads, and agricultural fields. As a consequence, there has been significant interest in the deployment of deep inference applications on microcontroller-scale devices [8] and internet-of-things endnodes [9]. This essentially requires to fit the tens of billions of operations of a net such as ResNet-18 [10] or Inception-v3/v4 [1] [11] on devices with a power budget of a few mW costing less than 1$ per device.

To meet these constraints, researchers have focused on reducing *i*) the *number of elementary operations*, with smaller DNNs [12] and techniques to prune unnecessary parts of the network [13]; *ii*) the *cost of an elementary compute operation*, by realizing more efficient software [8] and hardware [14] and lowering the complexity of elementary operations [15] [16]; and *iii*) the *cost of data movement*, again by reducing the size of DNNs and taking advantage of locality whenever possible [17].

An emerging trend to tackle *ii*) and *iii*) is that of fully binarizing both weights and activations in *Binary Neural Networks* (BNNs) [18] [19]. Their classification capabilities, together with the greatly reduced computational workload, represent a promising opportunity for integration in devices "at the edge", and even directly inside sensors [20]. Dropping the precision of weights and activations to a single bit enables the usage of simple XNOR operations in place of full-blown products, and greatly reduces the memory footprint of deep learning algorithms.

Software-based implementations of BNNs require special instructions for the popcount operation to be efficient and - more significantly - they require temporary storage of non-binary partial results either in the register file (with strong constraints on the final performance) or in memory (partially removing the advantage of binarization). In this paper, we contribute the design of the *XNOR Neural Engine* (XNE), a hardware accelerator IP for BNNs that is optimized for integration in a tiny microcontroller (MCU) system for edge computing applications. While being very small, it allows to overcome the limitations of SW-based BNNs and execute fast binarized convolutional and dense neural network layers while storing all partial results in its internal optimized buffer. We show that integrating the XNE within a MCU system leads to a flexible and usable accelerated system, which can reach peak efficiency of $21.6\,\mathrm{fJ}$ per operation but at the same time can be effectively used in real-world applications as it supports commonplace state-of-the-art BNNs such as ResNet-18 and ResNet-34 at reasonable frame rates ($>$8 fps) in less than 2.2 mJ per frame – a third of a millionth of the energy stored in an AAA battery. Finally, we show that even if binarization reduces the memory footprint and pressure with respect to standard DNNs, memory accesses and data transfers still constitute a significant part of the energy expense in the execution of real-world BNNs – calling for more research at the algorithmic, architectural and technological level to further reduce this overhead.

II. Related Works

The success of Deep Learning and, in particular convolutional neural networks, has triggered an exceptional amount of interest in hardware architects and designers who have tried to devise the most efficient way to deploy this powerful class of algorithms on embedded computing platforms. Given the number of designs that have been published for CNNs, we

This article will be presented in the International Conference on Hardware/Software Codesign and System Synthesis 2018 (CODES'18) and will appear as part of the ESWEEK-TCAD special issue. This work was partially supported by Samsung under the GRO project "SCAlable Learning-in-place Processor".

F. Conti and L. Benini are with the Integrated Systems Laboratory, D-ITET, ETH Zürich, 8092 Zürich, Switzerland and with the Energy-Efficient Embedded Systems Laboratory, DEI, University of Bologna, 40126 Bologna, Italy. P. D. Schiavone is with the Integrated Systems Laboratory, D-ITET, ETH Zürich, 8092 Zürich, Switzerland.

E-mail: {fconti,pschiavo,lbenini}@iis.ee.ethz.ch.

Dataset / Network	Top-1 Acc.	CONV / FC weights
MNIST / fully connected BNN [18]	99.04 %	- / 1.19 MB
SVHN / fully connected BNN [18]	97.47 %	139.7 kB / 641.3 kB
CIFAR-10 / fully connected BNN [18]	89.95 %	558.4 kB / 1.13 MB
ImageNet / ResNet-18 XNOR-Net [19]	51.2 %	1.31 MB / 2.99 MB
ImageNet / ResNet-18 ABC-Net M=3,N=3 [21]	61.0 %	3.93 MB / 8.97 MB
ImageNet / ResNet-18 ABC-Net M=5,N=5 [21]	65.0 %	6.55 MB / 14.95 MB
ImageNet / ResNet-34 ABC-Net M=1,N=1 [21]	52.4 %	2.51 MB / 2.99 MB
ImageNet / ResNet-34 ABC-Net M=3,N=3 [21]	66.7 %	7.54 MB / 8.97 MB
ImageNet / ResNet-34 ABC-Net M=5,N=5 [21]	68.4 %	12.57 MB / 14.95 MB

TABLE I: BNNs proposed in literature, along with the related top-1 accuracy and weight memory footprint.

will focus on a more direct comparison with accelerators that explicitly target a tradeoff between accuracy and energy or performance, keeping in mind that state-of-the-art accelerators for "conventional" fixed-point accelerators such as Orlando [22] are able to reach energy efficiencies in the order of a few Top/s/W.

The approaches used to reduce energy consumption in CNNs can be broadly categorized in two categories, sometimes applied simultaneously. The first approach is to prune some calculations to save time and energy, while performing the rest of the computations in "full precision". One of the simplest techniques is that employed by *Envision* [23] by applying Huffman compression to filters and activations, therefore saving a significant amount of energy in the transfer of data on- and off-chip. A similar technique, enhanced with learning-based pruning of "unused" weights, has been also proposed by Han et al. [13] and employed in the *EIE* [14] architecture. *NullHop* [24] exploits activation sparsity to reduce the number of performed operations by a factor of 5-10× (for example, up to 84% of input pixels are nil in several layers of ResNet-50).

The other popular approach is to drop the arithmetic precision of weights or activations, to minimize the energy spent in their computation. Up to now, this approach has proven to be very popular on the algorithmic side: DoReFaNet [15], BinaryConnect [25], BinaryNet [18] and XNOR-Net [19] have been proposed as techniques to progressively reduce the precision of weights and activations by quantizing it to less than 8 bits or outright binarizing it, at the cost of retraining and loss of accuracy. More recently, methods such as ABC-Net [21] and Incremental Network Quantization [26] have demonstrated that low-precision neural networks can be trained to an accuracy decreased $< 5\%$ with respect to the full precision one. Table I lists some of the BNNs proposed in the state-of-the-art, along with their accuracy and memory footprint. Naturally, this approach lends itself well to being implemented in hardware. The *Fulmine* SoC [9] includes a vectorial hardware accelerator capable of scaling the precision of weights from 16 bits down to 8 or 4 bits, gaining increased execution speed with similar power consumption. *Envision* [23] goes much further: it employs dynamic voltage, frequency and accuracy scaling to tune the arithmetic precision of its computation, reaching up to 10 Top/s/W. *YodaNN* [27] drops the precision of weights to a single bit by targeting binary-weight networks (activations use "full" 12-bit precision), and can reach up to 61 Top/s/W using standard cell memories to tune down the operating voltage.

To reach the highest possible efficiency, binary and ternary neural networks are perhaps most promising as they minimize the energy spent for each elementary operation, and also the amount of data transferred to/from memory, which is one of the biggest contributors to the "real" energy consumption. One of the first architectures to exploit these peculiarities has been *FINN* [28], which is able to reach more than 200 Gop/s/W on a Xilinx FPGA, vastly outperforming the state-of-the-art for FPGA-based deep inference accelerators. Recent efforts for the deployment of binary neural networks on silicon, such as *BRein* [29], *XNOR-POP* [30], *Conv-RAM* [31] and Khwa et al. [32] have mainly targeted in-memory computing, with energy efficiencies in the range 20-55 Top/s/W. However, the advantage of this methodology is not yet clear, as more "traditional" ASICs such as *UNPU* [33] and *XNORBIN* [34] can reach a similar level of efficiency of 50-100 Top/s/W. Finally, mixed-signal approaches [35] can reach 10× higher efficiency, with much steeper non-recurrent design and verification costs.

Our work in this paper tries to answer a related, but distinct question with respect to the presented state-of-the-art: how to design a BNN accelerator tightly integrated within a microcontroller (so that SW and HW can efficiently cooperate) – and how to make so while taking into account the system level effects related to memory which inevitably impact real-world BNN topologies such as ResNet and Inception. Therefore, we propose a design based on the tightly-coupled shared memory paradigm [36] and evaluate its integration in a simple, yet powerful, microcontroller system.

III. Architecture

A. Binary Neural Networks primer

In binary neural networks, inference can be mapped to a sequence of convolutional and densely connected layers of the form

$$\mathbf{y}(k_{out}) = \mathrm{bin}_{\pm 1}\left(\mathbf{b}_{k_{out}} + \sum_{k_{in}}\left(\mathbf{W}(k_{out}, k_{in}) \otimes \mathbf{x}(k_{in})\right)\right) \tag{1}$$

where $\mathbf{W}$, $\mathbf{x}$, $\mathbf{y}$ are the binarized ($\in \pm 1$) weight, input and output tensors respectively; $\mathbf{b}$ is a real-valued bias; $\otimes$ is the cross-correlation operation for convolutional layers and a normal product for densely connected ones. $\mathrm{bin}_{\pm 1}(\cdot)$ combines batch normalization for inference with binarization of the integer-valued output of the sum in Equation 2:

$$\mathrm{bin}_{\pm 1}(t) = \mathrm{sign}\left(\gamma\frac{t-\mu}{\sigma} + \beta\right) \tag{2}$$

where β, γ, μ, σ are the learned parameters of batch normalization.

```
for k_out in range(0, N_out):                    output feature maps loop
  for k_in in range(0, N_in):                    input feature maps loop
    for i in range(0, h_out):                    spatial rows loop
      for j in range(0, w_out):                  spatial columns loop
        y[k_out,i,j] = 0
        for u_i in range(0, fs):                 filter rows loop
          for u_j in range(0, fs):               filter columns loop
            y[k_out,i,j] += W[k_out,k_in,u_i,u_j]
                          * x[k_in,i+u_i,j+u_j]
```

Listing 1: Baseline loops of a BNN convolutional layer[1].

A more convenient representation of the BNN layer can be obtained by mapping elements of value $+1$ to 1-valued bits and those of value -1 to 0-valued bits, and moving the bias inside the binarization function. Equation 2 can be reorganized into

$$\mathrm{bin}_{0,1}(t) = \begin{cases} 1 \text{ if } t \geq -\kappa/\lambda \doteq \tau, \text{ else } 0 \text{ (when } \lambda > 0) \\ 1 \text{ if } t \leq -\kappa/\lambda \doteq \tau, \text{ else } 0 \text{ (when } \lambda < 0) \end{cases} \tag{3}$$

where $\lambda \doteq \gamma/\sigma$, $\kappa \doteq \beta + \gamma/\sigma(b - \mu)$, and $\tau \doteq -\kappa/\lambda$ is a threshold defined for convenience in Section III-C3. Multiplications in Equation 1 can be replaced with XNOR operations, and sums with popcounting (i.e., counting the number of bits set to 1):

$$\mathbf{y}(k_{out}) = \mathrm{bin}_{0,1}\left(\sum_{k_{in}}\left(\mathbf{W}(k_{out}, k_{in}) \otimes \mathbf{x}(k_{in})\right)\right) \tag{4}$$

B. XNE operating principles

The XNOR Neural Engine we propose in this work has been designed to be able to execute both binarized convolutional and binarized dense layers. Convolutional layers consist of six nested loops on output feature maps, input feature maps, two output spatial dimensions, and two filter spatial dimensions; Listing 1 shows a naïve implementation of a convolutional layer in Python pseudo-code. Densely connected layers can be considered as a limit case of the convolutional layer for a 1×1 filter on a single pixel in the spatial dimensions, i.e. `h_out=w_out=fs=1`.

In modern topologies [10] [11], deeper convolutional layers have `N_out,N_in>h_out,w_out`; in other words, layers become "less convolutional" and more similar to densely connected layers. This leads towards choosing an architecture where pixel- or feature map-level parallelism is favored over filter-level parallelism (contrary to designs based on sliding windows). This is particularly true for BNNs, where energy efficiency can be attained only by operating on tens/hundreds of binary pixels in parallel – which cannot be done with filter-level parallelism on deeper layers.

A second fundamental consideration is that, since intermediate values of the *popcount* operation are integer, it is highly preferable to perform the operation of Equation 4 without storing them in memory. In other words, the accelerator has to be weight- and output-stationary [28] or input- and output-stationary. In the remainder on this paper, we focus exclusively on the latter case, although the XNE can arguably be used in both modes by swapping the roles of weights and inputs.

[1]The $\star$ and += operators indicate XNOR and popcount-accumulation respectively.

```
for i in range(0, h_out):                        spatial rows loop
  for j in range(0, w_out):                      spatial columns loop
    for k_out_major in range(0, N_out/TP):       output feature maps outer loop
      for k_out_minor in range(0, TP):
        k_out = k_out_major*TP + k_out_minor
        y[k_out,i,j] = 0
      for u_i in range(0, fs):                   filter rows loop
        for u_j in range(0, fs):                 filter columns loop
          for k_in_major in range(0, N_in/TP):   input feature maps outer loop
            for k_out_minor in range(0, TP):     output feature maps tile loop
              for k_in_minor in range(0, TP):    input feature maps tile loop
                k_out = k_out_major*TP + k_out_minor
                k_in  = k_in_major*TP  + k_in_minor
                y[k_out,i,j] += W[k_out,k_in,u_i,u_j]
                              * x[k_in,i+u_i,j+u_j]
```

Listing 2: Reordered DNN layer loops; the innermost loops (highlighted in light blue) are hardwired in the XNE engine, while the others can be implemented in the XNE microcode. Remainder loops are left out for simplicity.

We designed the XNE around a lean hardware engine focused on the execution of the feature loops of Listing 1. We execute these as hardwired inner loops, operating in principle on a fixed-sized input tiles in a fixed number of cycles[2]. A design-time *throughput parameter* (`TP`) is used to define the size of each tile, which is also the number of simultaneous XNOR operations the datapath can execute per cycle; every `TP` cycles, the accelerator consumes one set of `TP` input binary pixels and `TP` sets of `TP` binary weights to produce one set of `TP` output pixels.

Listing 2 shows how the convolutional layer is reorganized: *i*) the loops are reordered, bringing spatial loops to the outermost position, feature-map loops to the innermost position and filter loops in the middle; *ii*) the two feature loops are tiled and therefore split in a *tile* loop (cycling on a tile of `TP` iterations) and an *outer* loop (cycling on `nif/TP` or `nof/TP` tiles); *ii*) the output feature maps outer loop is moved outwards with respect to the filter loops. If `nif` and/or `nof` are not whole multiples of `TP`, "remainder" iterations have to be executed; these are left out of the listing for the sake of brevity. The innermost loops, which are shown highlighted in blue, are hardwired in the engine datapath as previously introduced and fully explained in Section III-C3, which details the datapath micro-architecture. The outermost loops, instead, are implemented by means of a tiny microcode processor embedded in the XNOR Neural Engine, as detailed in Section III-C2.

C. Accelerator architecture

Figure 1 shows the overall architecture of the XNE. The *controller*, which can be targeted in a memory-mapped fashion via a target port using the AMBA APB protocol, is responsible of coordinating the overall operation of the accelerator. It contains a latch-based register file, a central controller finite-state machine (FSM), and a *microcode processor* (further detailed in Section III-C2) that is responsible of implementing the outer loops of Listing 2. The *engine* contains the streaming datapath, which executes the inner loop operation of Listing 2. It operates on streams that use a simple valid-ready handshake similar to that used by AXI4-Stream [37]. Finally, the *streamer* acts as a transactor between the streaming domain used by

[2]The XNE can actually be configured to operate on smaller tiles when it is necessary, with a proportional decrease in loop latency.

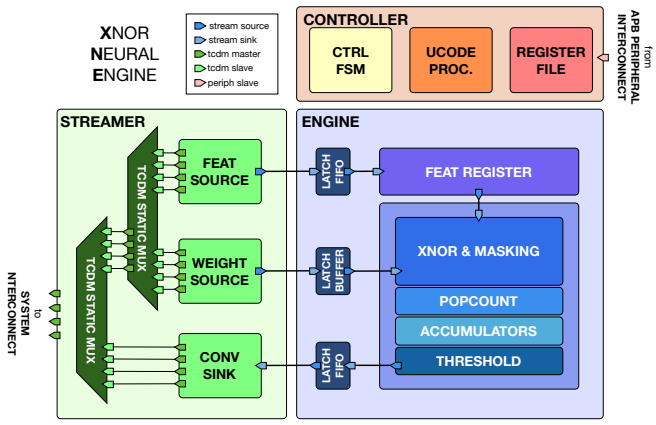


Fig. 1: XNOR Neural Engine overall architecture for TP=128.

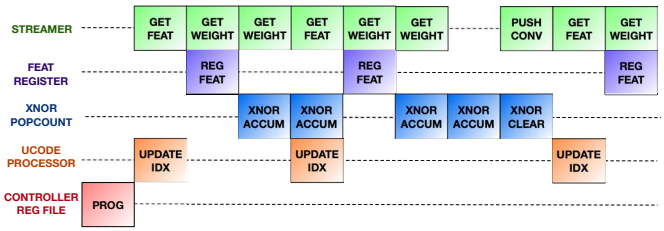


Fig. 2: Example of XNE operation divided in its main phases.

```
loop_stream_inner:  # for k_in_major in range(0, N_in/TP)
  - { op : add, out : W,       in : TPsquare }        W ← W + TP²
  - { op : add, out : x,       in : TP       }        x ← x + TP
loop_filter_x:      # for u_j in range(0, fs)
  - { op : add, out : W,       in : nif      }        W ← W + nif
  - { op : add, out : x,       in : nif      }        x ← x + nif
loop_filter_y:      # for u_i in range(0, fs)
  - { op : mv,  out : x,       in : x_major  }        x ← x_major
  - { op : add, out : x,       in : w_X_nif  }        x ← x + width × nif
loop_stream_outer:  # for k_out_major in range(0, N_out/TP)
  - { op : add, out : W,       in : TPsquare }        W ← W + TP²
  - { op : add, out : y,       in : TP       }        y ← y + TP
loop_spatial_x:     # for j in range(0, w_out)
  - { op : add, out : y,       in : nof      }        y ← y + nof
  - { op : add, out : x_major, in : nif      }        x_major ← x_major + nif
  - { op : mv,  out : W,       in : zero     }        W ← 0
  - { op : mv,  out : x,       in : x_major  }        x ← x_major
loop_spatial_y:     # for i in range(0, h_out)
  - { op : add, out : y,       in : nof      }        y ← y + nof
  - { op : add, out : x_major, in : nif      }        x_major ← x_major+nif
```

Listing 3: Microcode specification for the six loops shown in Listing 2. `W`, `x`, `y` and `x_major` are mnemonics for the four R/W registers; `TPsquare`, `TP`, `nif`, `nof`, `w_X_nif`, `ow_X_nof`, `zero` are mnemonics for the R/O registers used in this implementation.

the internal engine and the memory system connected to the accelerator. It is capable of transforming streams of width multiple of 32 bits into byte-aligned accesses to the cluster shared memory, and vice versa.

Figure 2 shows a high-level view of how the XNE operates. The controller register file is first programmed with the DNN layer parameters (e.g. `nif`, `nof`, `fs`, etc.) and with the microcode byte code. The central controller FSM then orchestrates the operation of the XNE, which is divided in three phases: FEATURE LOADING, ACCUMULATION, THRESHOLDING/BINARIZATION. In the FEATURE LOADING phase, the i-th feature TP-vector is loaded from the streamer, while at the same time the microcode processor starts updating the indeces used to access the next one. In the ACCUMULATION, for TP iterations a new weight TP-vector is loaded and multiplied by the feature vector, and the result is saved in an accumulator. In the THRESHOLDING AND BINARIZATION phase, TP threshold values are loaded from memory and used to perform the binarization, then the binarized outputs are streamed out of the accelerator. These three phases are repeated as many times as necessary to implement the full loop of Listing 2.

1) Interface modules: The interface that the XNE exposes follows the paradigm of shared-memory, tightly coupled Hardware Processing Engines [36]. The XNE has a single APB target port, which allows memory mapped control of the XNE and access to its register file, and TP/32 master ports (each 32 bits wide) enabling access to the shared memory system via word-aligned memory accesses. Finally, a single *event* wire is used to signal the end of the XNE computation to the rest of the system.

The *controller* module, which is the direct target of the slave port, consists of the memory-mapped register file, a finite-state machine used to implement the main XNE operation phases as shown in Figure 2, and a microcode processor to implement the loops in Listing 2 (as described in Section III-C2). The memory-mapped register file uses standard cell memories implemented with latches to save area and power with respect to conventional flip-flops. It includes two sets of registers: *generic* ones, used to host parameters that are assumed to be static between the execution of multiple jobs, and *job-dependent* ones, for parameters that normally change at every new job (such as base pointers). The latter set of registers is duplicated so that one new job can be offloaded from the controlling processor to the XNE even while it is still working on the current one.

The *streamer* module contains the blocks necessary to move data in and out of the accelerator through its master ports, and transform the memory accesses into coherent streams to feed the accelerator inner engine[3]. These are organized in separate hardware modules, two *sources* for incoming streams (one for weights/thresholds, one for input activations) and one *sink* for the outgoing one (output activations). Both the two sources and the sink include an own address generation block to start the transaction in memory and a realigner to transform vectors that start from a non-word-aligned base into well-formed streams, without assuming that the memory system outside of the accelerator can natively support misaligned accesses. The memory accesses produced by the source and sink modules are mixed by two static mux/demux blocks; the controller FSM ensures that only one is active at any given cycle and that no transactions are lost.

2) Microcode processor: Instead of relying on an external processor to compute the new offsets for memory access, to iterate the inner loop execution, and to maintain overall state, the XNE can use a small internal *microcode processor* to implement the six nested outer loops shown in Listing 2. The microcode processor has four R/W registers, used to compute the `i`,`j`, `k_out_major`, `u_i`, `u_j`, `k_in_major` indeces of Listing 2; and can access sixteen R/O registers. The latter are used to store loop ranges and iteration values, coming from the register file directly or indirectly, i.e. computed

[3] Controller and streamer IPs are available as open-source at `github.com/pulp-platform/hwpe-ctrl` and `github.com/pulp-platform/hwpe-stream` respectively.

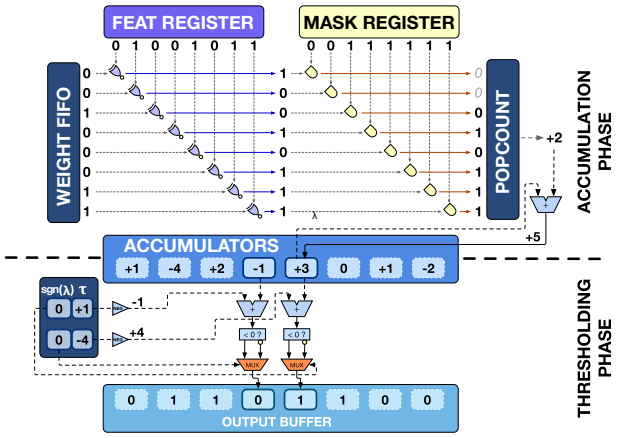


Fig. 3: XNE datapath for XNOR, popcounting, accumulation and thresholding (TP=8).

from register file values using simple sequential multipliers to minimize hardware overhead.

The microcode processor uses a custom tiny ISA with two "imperative" instructions, *ADD* (add/accumulate) and *MV* (move). They use one of the R/W registers as output and one R/O or R/W register as input; the *ADD* instruction implicitly uses the output register as a second input. The microcrode ISA also includes one "declarative" *LOOP* instruction, containing the iteration range of each loop and the base address and number of micro-instructions associated to it. The hardware implementation of this ISA is a single-stage execution pipeline controlled by four simple finite-state machines operating in lockstep; they compute the address of the next micro-instruction to execute, its index within the current loop, the next iteration index of the current loop, and the next loop to be taken into account.

The microcode associated to the functionality presented in Listing 2 (six loops) occupies 28B in total (22B for the imperative part, 6B for the declarative one) which are mapped directly within the XNE register file. The final microcode, which is specified in a relatively high-level fashion by means of a description in the YAML markup language, can be seen in Listing 3. This description can be compiled into a bitstream using a simple Python script and added to the preamble of an application; the microcode is stored in the "generic" section of the register file and is kept between consecutive jobs unless explicitly changed.

3) Datapath micro-architecture: The XNE datapath is composed by the blocks responsible of performing vector binary multiply (realized by means of XNOR gates), accumulation (within a latch-based register file) and thresholding to determine normalized binary outputs. The datapath is fed with the weight/threshold and the input activation streams coming from the streamer sources through two-element FIFOs; it produces an output activation stream into a decoupling two-element FIFO, which on turn is connected with the streamer sink. Figure 3 illustrates the structure of the datapath in a case where TP is 8. The input feature TP-vector is stored in a *feature register* to be reused for $\min(\texttt{TP},\texttt{N\_out})$ cycles (one for each accumulator used). Once an output feature vector has been produced by the XNE datapath, it is completely computed and never used again. With the microcoding strategy proposed in Listing 3, a single input feature vector has to be reloaded $\texttt{fs}^2$ times, and afterwards it is completely consumed.

The weight TP-vector stream produced by the streamer is decoupled from the main datapath by means of a four-element FIFO queue; at each cycle in the main binary convolution execution stage, the feature vector is "multiplied" with the weight stream by means of TP XNOR gates, producing the binary contributions of all TP input feature elements to a single output feature element. These contributions are masked by means of an array of AND gates to allow the XNE to work even when the number of input features is smaller than TP. A combinational reduction tree is used to perform the *popcount* operation, i.e. to count the number of 1's in the unmasked part of the product vector. The output is accumulated with the current state of an accumulator register; there are in total TP accumulators, one for each output computed in a full accumulation cycle. Accumulated values are computed with 16 bit precision and saturated arithmetic.

To implement the binarization function of Equation 3, the value stored in the accumulators is binarized after a thresholding phase, which encapsulates also batch normalization. The binarization thresholds are stored in a vector of TP bytes, and loaded only when the accumulated output activations are ready to be streamed out. Each byte is composed of 7 bits (one for sign, six for mantissa) representing τ, plus 1 bit used to represent $\text{sign}(\lambda)$ (used to decide the sign of the comparison). The 7-bit τ is left-shifted of a configurable amount of bits S_τ, to enable the comparison with the 16-bit accumulators. The output of the thresholding phase is saved in a FIFO buffer, from which it is sent to the streamer module (see Figure 1) so that it can be stored in the shared memory.

4) Impact of accumulator and threshold truncation: According to our experiments, the impact of truncating accumulators (to 16 bits) and thresholds (to 7 bits) is very small. Errors due to accumulator truncation can happen only on bigger layers than what is found in most BNN topologies (e.g., even a layer with nif=1024, fs=5 does not have enough accumulations per output pixel to hit the accumulator dynamic range), and only in consequence of unlikely imbalances between 0's and 1's; saturation provides a mitigation mechanism for many of these cases.

For what concerns the truncation of batch-normalization thresholds to 7 bits, if a shift $S_\tau > 0$ is being used, a superset of the accumulator values that could be affected (i.e. that could be binarized incorrectly) is given by the worst-case error interval $\left[\tau \pm 2^{S_\tau - 1}\right]$. The probability that accumulator values reside within this interval (i.e., they are near the threshold between providing a +1 or -1) depends on the layer size and the training methodology, as well as the actual input of the BNN. In our experiments of Section IV-B3 using the training method of Courbariaux et al. [18], we did not observe any accuracy degradation with S_τ values (between 0 and 2) adequate to represent all the dynamic range of the thresholds.

IV. Experimental Results

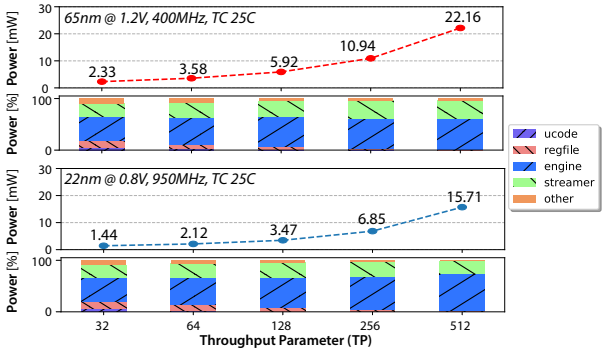
In this section, we evaluate the energy and area efficiency of the proposed XNE accelerator design taken "standalone" with several choices of the TP parameter; then we showcase and evaluate a full microcontroller system augmented with the XNE accelerator.

TP	32	64	128	256	512
µcode	**5.2** (12%)	**5.2** (9%)	**5.2** (6%)	**5.2** (3%)	**5.2** (1%)
regfile	**16.8** (40%)	**16.8** (29%)	**16.8** (18%)	**16.8** (10%)	**16.8** (5%)
engine	**11.8** (28%)	**21.8** (37%)	**41.2** (44%)	**82.8** (49%)	**165.9** (52%)
streamer	**8.5** (20%)	**15.2** (25%)	**29.5** (32%)	**63.2** (38%)	**133.9** (42%)

(a) Area in 10^3 µm^2 in 65nm technology (target 300MHz@1.2V WC 125C).

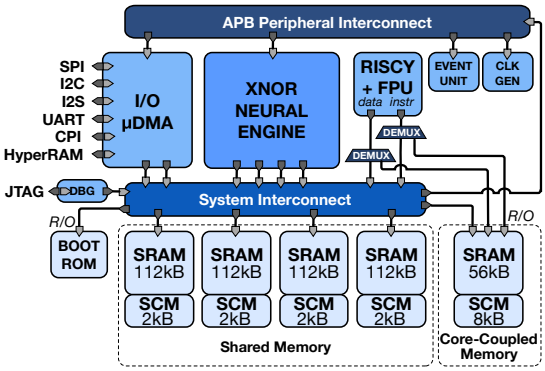
TP	32	64	128	256	512
µcode	**0.8** (11%)	**0.8** (7%)	**0.8** (5%)	**0.8** (3%)	**0.8** (1%)
regfile	**3.0** (40%)	**3.0** (29%)	**3.0** (18%)	**3.0** (10%)	**3.0** (6%)
engine	**2.2** (29%)	**4.0** (39%)	**7.6** (47%)	**15.7** (54%)	**31.9** (57%)
streamer	**1.5** (20%)	**2.6** (25%)	**4.8** (30%)	**9.7** (33%)	**20.2** (36%)

(b) Area in 10^3 µm^2 in 22nm technology (target 800MHz@0.8V WC 125C).

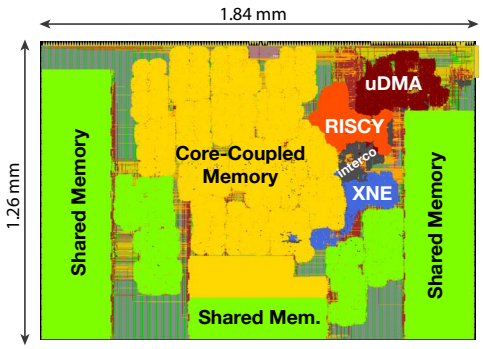


(c) Power in nominal operating point in ACCUMULATION phase.

Fig. 4: Stand-alone XNE results in terms of area and power in nominal operating conditions for the two target technologies.



(a) MCU architecture.



(b) MCU floorplan in 22nm.

Fig. 5: Architecture of the microcontroller system (MCU) and its layout in 22nm technology.

A. Standalone XNE

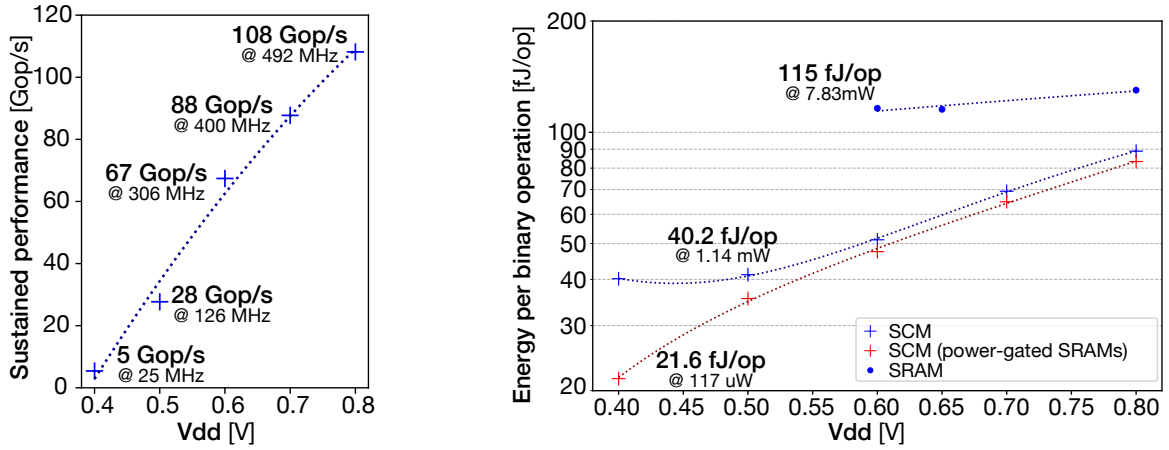
The main architectural parameter of the XNE, the *throughput parameter* TP, can be used to choose the amount of hardware parallelism exploited by the accelerator, and the related required number of master ports on the memory side. In this section, we make a first evaluation on how changing this parameter can influence the area and power of the accelerator. We implemented the XNE in synthesizable SystemVerilog HDL using TP as a design-time parameter, sweeping from TP=32 to TP=512 in geometric progression.

The various versions of the XNE were synthesized using Synopsys Design Compiler 2017.09 targeting 300 MHz@0.59V, 125C and 800 MHz@1.08V, 125C in 65nm and 22nm, respectively (in worst case). Afterwards, we performed a place & route run of the block using Cadence Innovus 16.10. We targeted 65% utilization on a square area; as the XNE is synthesized stand-alone instead of in coupling with a multi-banked memory, this P&R does not accurately model all effects present when deploying an XNE in a real platform. However, it enables vastly more accurate power prediction with respect to post-synthesis results after clock tree synthesis and the extraction of wiring parasitics. Moreover, the 65% utilization target is conservative enough so that it is possible to check that the XNE does not introduce congestion when routed on a more realistic design For power estimation, performed with Synopsys PrimeTime PX 2016.12, we used activity dumps from post-layout simulation and we targeted the typical corner. After P&R, all XNEs are able to work at up to 400 MHz@1.25V, 25C (in 65nm) / 950 MHz@0.72V, 25C (22nm) in the typical corner.

In Figure 4, we report the area of the synthesized XNE with the 65nm and 22nm libraries; the Table shows that the fixed costs of the microcode processor and register file are progressively absorbed as the size of the engine and streamer increase near-linearly with TP. Figure 4 also reports power estimation results in nominal operating conditions from the various versions of the XNE (in the active ACCUMULATION phase), shows similar scaling, with the engine and streamer modules being responsible for most of the power consumed by the XNE. The latter point indicates that, as expected, the XNE shows a high internal architectural efficiency.

B. XNE in a MCU System

The XNE is designed as a tightly-coupled accelerator engine [36] and it can be more completely evaluated when integrated within a full system-on-chip. To this end, given the results shown in Section IV-A, we selected the design with TP= 128 for integration in a HW-accelerated microcontroller system (MCU). The MCU uses the RISCY [38] RISC-V ISA core and features also an autonomous I/O subsystem (uDMA) [39], capable of moving data from/to memory and to/from selected I/O peripherals (SPI, I2C, I2S, UART, CPI, and HyperRAM) - and also of marshaling data in the shared



(a) Performance in Gop/s.

(b) MCU-level energy efficiency in fJ/op.

Fig. 6: Sustained performance and MCU system level energy efficiency, when using the XNE to execute binary convolutions on data stored in SRAM or SCM memories. Dotted lines are used for curve fitting between characterized operating points (crosses / circles).

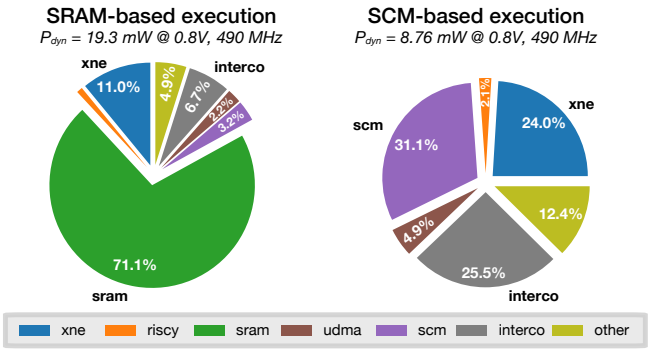


Fig. 7: Distribution of dynamic power, when using the XNE to execute binary convolutions on data stored in SRAM or SCM memories.

memory[4]. We targeted the 22nm technology referred in Section IV-A; we used the same tools reported in Section IV-A for synthesis and backend.

Figure 5 shows the architecture of the MCU system and its floorplan, where the most relevant blocks have been highlighted. The MCU is internally synchronous and memories, core and accelerator belong to a single clock domain. The MCU has 64 kB of core-coupled memory accessed prioritarily by RISCY and 456 kB of memory shared between RISCY, uDMA and XNE. Both kinds of memory are hybrids of SRAM banks and latch-based standard-cell-memory [27] (SCM). Specifically, 8 kB of core-coupled memory are made of multi-ported SCMs and 8 kB of shared memory are single-ported SCMs. As will be detailed in the following of this section, SCMs are essential to keep the MCU operational below the rated operating voltage for SRAM memories, and they are also typically more energy-efficient than SRAMs, although they are much less area-efficient. Finally, all SRAMs operate on a separate power domain and can be completely turned off by an external DC-DC converter.

1) Performance evaluation: To evaluate the performance of the XNE, we compare with an efficient software implementation targeted at low-power microcontrollers [40]. A naive implementation of the binary convolution kernel requires on average 2 cycles per each *xnor-popcount*, which is clearly highly inefficient due to the extremely fine granularity of the operation. By performing multiple convolutions on adjacent pixels in a parallel fashion, and the RISCY instructions for popcount, throughput can be increased by $\sim 9\times$ up to 3.1 op/cycle[5].

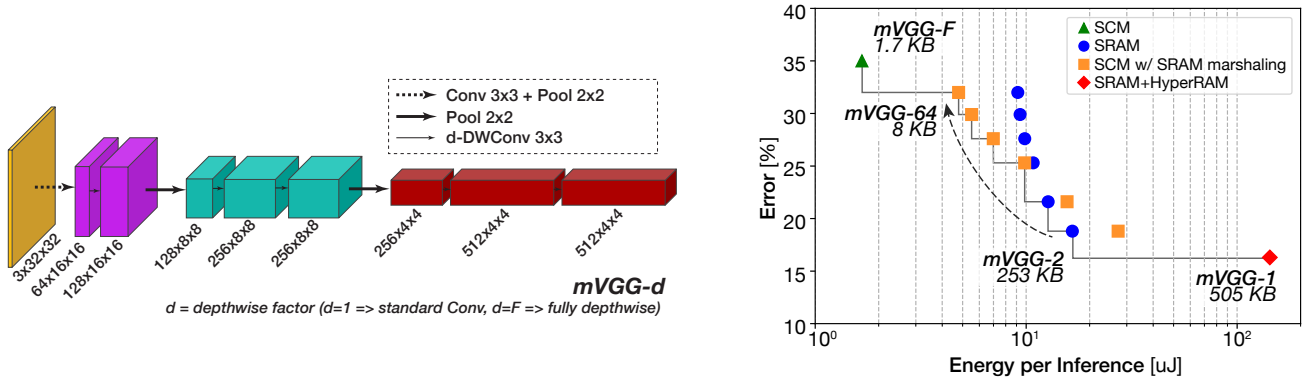
On the other hand, the XNE integrated in the MCU system can sustain a throughput of 220 op/cycle under normal conditions (86% of its theoretical peak throughput with TP=128, with the drop being caused by memory contention and small control bubbles). This means that the XNE can provide a net improvement of $71\times$ to throughput for binary convolutions and densely connected layers with respect to optimized software. Figure 6a shows the overall sustained throughput at the MCU system level in various operating points in typical conditions, with operating frequency extracted from PrimeTime timing analysis. At the nominal operating point ($0.8\,\mathrm{V}$), the MCU works at up to $490\,\mathrm{MHz}$ and the XNE can reach a throughput of up to 108 Gop/s.

2) Energy efficiency evaluation: We evaluated separately the power consumption of the XNE when insisting on the SRAMs, which are rated for operation between $0.6\,\mathrm{V}$ and $0.8\,\mathrm{V}$, and on the SCMs, which we evaluated down to $0.4\,\mathrm{V}$. Since SRAMs can be entirely switched-off externally, and the MCU does not depend on them for essential operations, we evaluated both the case in which they are fully switched off and the one in which they are simply not used (and therefore they consume static leakage power).

Figure 7 shows the outcome of this evaluation in terms of dynamic power at $0.8\,\mathrm{V}$, while executing an XNE-based binary convolution kernel either on data located on SRAM

[4] The MCU is based on a modified version of PULPissimo (`github.com/pulp-platform/pulpissimo`), which includes RISCY, uDMA and an example accelerator.

[5] Throughout the paper, we count xnor and popcount as separate operations, therefore 1 xnor + 1 popcount = 2 op



(a) *mVGG-d* topology based on Courbariaux et al. [18].

(b) Energy per inference vs Error.

Fig. 8: mVGG binary neural network energy per inference vs error trade-off on *mVGG-D*; in the rightmost plot, green triangles, blue circles, orange squares and red diamonds represent respectively usage modes on pure SCM @0.4V, on SRAM / on SCM with SRAM marshaling @0.6V, and with HyperRAM marshaling @0.6V. The grey solid line indicates the Pareto frontier.

or on SCM. When executing on the SRAM, the dynamic power due to memory clearly dominates over computation, by a factor of 7.1×, taking into account also the power spent in the system interconnect. Conversely, SCM-based execution is more balanced, as SCMs consume ∼3× less then SRAMs. In both cases, memory accesses are largely due to weights, which are loaded many times and used only once in the XNE design.

The advantage of working on SCMs is clearer when we evaluate energy efficiency in terms of femtoJoules per operation, as shown in Figure 6b. There is a factor of ∼ 2-3× between SRAM- and SCM-based execution, especially when the operating voltage is reduced[6]. SCMs, which are ∼2-3× less power-hungry and do not stop working at low voltage, enable the XNE to deliver much better energy efficiency. If we do not fully switch down the SRAMs, the minimum energy point is located near the 0.5 V operating point, where the MCU delivers 28 Gop/s and 40.2 fJ per operation are required - equivalent to a system-level efficiency of 25 Top/s/W. Power-gating the SRAMs vastly reduces leakage power and moves the minimum energy point further down in operating voltage: at 21.6 fJ per operation at 0.4 V.

3) Energy-accuracy tradeoff in BNNs: The most efficient use case for the MCU platform is clearly when entire network topologies can be fully deployed on the shared memory, and in particular on the SCM. To fully showcase the impact of the model memory footprint on the overall efficiency, we used a simple topology derived from a reduced version of the popular VGG [41], as proposed by Courbariaux et al. [18]; we trained it on the CIFAR-10 dataset for 150 epochs using their same binarization strategy, ADAM optimizer, and initial learning rate 0.005. Figure 8a shows the *mVGG-d* network. To scale the number of parameters stored in memory in a smooth fashion, we kept the network architecture of *mVGG-d* fixed, but progressively modified the nature of convolutional layers from the standard definition of 1 in the direction of depthwise separable convolutions [42] following the parameter d. Specifically, we modeled convolutions of the form

$$\mathbf{y}(k_{out}) = \text{bin}_{\pm 1}\left(\sum_{k_{in}=d\cdot k_{out}}^{(d+1)\cdot k_{out}-1}\left(\mathbf{W}(k_{out},k_{in})\otimes\mathbf{x}(k_{in})\right)\right) \quad (5)$$

This model is fully supported by the XNE with minor microcode modifications.

To model power consumption in the various versions of *mVGG-d*, we consider several usage modes. When the network (parameters and partial results) fully fits within the shared SCM memory, we operate at the most efficient energy point – 0.4 V with power-gated SRAMs, consuming 21.6 fJ per operation. Conversely, when it does not fit the SCMs but fits in the SRAMs, we operate at 0.6 V, consuming 115 fJ per operation. As an alternative, we also support a mode in which weights, which are responsible for the majority of the energy consumption, are marshaled from SRAM to a temporary SCM-based buffer. In this case, the energy cost of computation is reduced to 52 fJ, but there is an overhead of ∼8.7 pJ per bit to move weights from SRAM to SCM. Finally, when the SRAM is too small to host the weights, they are stored in an external memory and loaded to the SRAM when needed by means of the uDMA. In this case, we considering using a low-power Cypress HyperRAM 8MB DRAM memory [43] as external memory, directly connected to the MCU uDMA. The HyperRAM operates at 125 MHz (1 Gbit/s) and 28.6 pJ per bit read.

Figure 8b shows the results of this evaluation in terms of the Pareto plot of the size/energy versus accuracy trade-off in *mVGG-d* BNNs. We scale d with power-of-two values from 1 to 64 and consider also the case of fully depthwise separable convolutions (*mVGG-F*). The results clearly show the impact of memory energy on even small benchmarks such as *mVGG-d*. The most accurate model, *mVGG-1*, is only ∼6% from the current state-of-the-art for BNNs on CIFAR-10 [18]; however, this model consumes roughly 10x of *mVGG-2*, because it cannot run at all without the external HyperRAM. Increasing d, we observe that the energy penalty of marshaling data from SRAMs to SCMs is increasingly reduced up to a point (*mVGG-8*) where it becomes less significant than the cost of operating directly on the SRAMs; hence it becomes convenient

[6]According to the SRAM model we used, the internal power which dominates in SRAMs is less dependent on Vdd than the net switching power which dominates in most other modules – this is also the reason for which the energy efficiency in SRAM mode is flatter in Figure 6b.

to marshal data between the two. Finally, the *mVGG-F* model is so small that it can be run entirely on SCMs and consumes $100\times$ less than *mVGG-d*, but it suffers a significany penalty in terms of accuracy.

4) Real-world sized BNN execution: The size of real-world state-of-the-art DNN topologies for most interesting problems is such that it does not make sense at all to consider fully localized execution on the 520 kB of on-chip memory of the MCU system, even with BNNs. Supporting execution aided by external platforms is, therefore, critical. To minimize the continuous cost that would be implied by transfer of partial results, we dimensioned the MCU system so that relatively big BNN topologies can be run using the external memory exclusively for storing weights.

As representatives of real-world sized BNNs, we chose ResNet-18 and ResNet-34 [10], which can be fully binarized providing a top-5 accuracy of 67.6% and 76.5% respectively on the ImageNet database [21]. A binarized implementation of the ResNets requires $128\,\mathrm{kB}$ for input, output and partial results buffering (taking into account also shortcut connections), plus a maximum of $288\,\mathrm{kB}$ for the weights of a single layer; the final densely connected layer requires more memory, but it has an extremely small footprint for partial result buffering, and therefore it is possible to efficiently divide the computation in filtering tiles executed sequentially. Overall, it is possible to execute both these topologies on the tiny XNE-equipped MCU system without any energy cost for moving partial results.

To evaluate how efficient the deployment of such a model can be, we consider the same system of Section IV-B3, with an 8 MB HyperRAM connected to the uDMA. We consider the SRAM-based execution mode for this evaluation. We consider weights to be transferred asynchronously by means of the uDMA, performing double buffering to overlap memory access by the XNE with the fetching of the next set of weights. ResNet-18 and ResNet-34 require 3.64×10^9 and 7.34×10^9 operations respectively. In this operating mode, the compute time dominates for all layers except the last group of convolutions and the final fully connected layer in both ResNet-18 and ResNet-34. ResNet-18 inference can be run at $\sim$14.7 fps, spending $1.45\,\mathrm{mJ}$ per frame on a standard 224×224 input; for the latter at 8.9 fps, spending $2.17\,\mathrm{mJ}$ per frame.

In both cases, the contribution of memory traffic to energy consumption is dominant, mostly due the final layers (especially the fully connected one, which is memory-bound). The impact of these layers is more relevant in ResNet-18 than in ResNet-34, hence memory traffic energy is more dominant in the former case (by $2.5\times$) than in the latter (by 60%). Even if the cost of memory traffic cannot be entirely removed, the design of the MCU system mitigates this cost by making most data movements unnecessary, as weights are directly loaded on the shared SRAM and partial results never have to leave it.

5) Comparison with the state-of-the-art and discussion: Table II shows a comparison between our work and the current state-of-the-art in hardware accelerators for Binary Neural Networks. Contrary to our solution, current systems do not implement a full microcontroller or System-on-Chip, but consist either in near-memory computing techniques (*BRein*, *XNOR-POP*) or dedicated ASICs for binary neural networks.

Of all the ASIC accelerators taken into account, Bankman et al. [35] claims by far the highest energy efficiency (more than 700 Top/s/W), but they are dependent on full-custom mixed signal IPs that are known to be delicate with respect to on-chip variability and difficult to port between technologies. Moreover, their approach has hardwired convolution size (2×2), which severely limits their flexibility to implement different kinds of convolutions.

XNORBIN [34] achieves the second-best result with a much more traditional fully-digital ASIC architecture, achieving almost 100 Top/s/W with a 65nm chip. Compared with our MCU design, the main advantage of *XNORBIN* is placed in its custom memory hierarchy, enabling a non-constrained design for what concerns the accelerator core. This fact accounts for most of its advantage in terms of raw energy efficiency. However, *XNORBIN* does not include enough memory to implement BNNs bigger than AlexNet and, in general, it does not have facilities to enable exchange of data with the external world. Similarly, *UNPU* [33] targets efficient execution without particular attention to communication. It is roughly $16\times$ bigger than XNORBIN, but reaches only half the energy efficiency.

Compared to *UNPU* and *XNORBIN*, the best fully digital designs currently in the state-of-the-art (to the best of our knowledge), our work tackles a different problem: not providing the lowest energy solution as-is, but a methodology and an accelerator IP for the integration of BNNs within a more complete System-on-Chip solution, with an eye to system level problems, in particular the cost of memory accesses. The XNE has been designed to make efficient use of the relatively limited memory bandwidth allowed in an MCU-like SoC (the interfaces are active $\sim$95% of the overall execution time in many cases) and to be small and unobtrusive in terms of area ($\sim$1.5% of the proposed MCU) and timing closure (30% shorter critical path than the overall MCU system). Conversely, the design of an ASIC accelerator deals with different architectural constraints – in particular, the memory hierarchy is designed around the accelerator to provide the maximum effective memory bandwidth. For example, *XNORBIN* uses an ad-hoc memory hierarchy in which weights, feature maps and lines are stored separately (the datapath is fed by a linebuffer) amounting for improved effective memory bandwidth available with respect to our design (and hence higher efficiency), at the expense of flexibility and of area.

To the best of our knowledge, the XNE-accelerated MCU is the only design that can execute *software-defined* BNNs in an efficient way, by taking advantage of the tight integration between the XNE accelerator, the RISCY core and the uDMA to speed up nested loops of binary matrix-vector products. The generality of this mechanism makes the MCU capable of dealing with all BNNs in which the linear part of convolutional and fully connected layers is constituted of binary matrix-vector products (a group which contains most known neural network topologies), provided that the external memory can store all weights.

V. Conclusion

To the best of our knowledge, this paper is the first to introduce a fully synthesizable ultra-efficient hardware accelerator IP for binary neural networks meant for integration within microcontroller systems. We also propose a microcontroller system (MCU) designed to be flexible and usable in many application scenarios, but at the same time extremely efficient (up to 46 Top/s/W) for BNNs. The MCU is the only work in the current state-of-the-art capable of executing real-world sized BNN topologies such as ResNet-18 and ResNet-34; the

Name	Technology	Maturity	Core Area [mm^2]	Peak Perf. [Top/s]	Energy Eff. [Top/s/W]	On-chip Mem. [kB]
BRein [29]	65nm	silicon	3.9	1.38	6	-
XNOR-POP [30]	32nm	layout	2.24	∼5.7	∼24	512
UNPU [33]	65nm	silicon	16	7.37	51	256
XNORBIN [34]	65nm	layout	1.04	0.75	95	54
Bankman et al. [35]	28nm mixed-signal	silicon	4.84	-	722	329
This work (MCU, SCM w/ SRAM off)	**22nm**	**layout**	**2.32**	**0.11**	**46**	**520**
This work (MCU, SCM)	**22nm**	**layout**	**2.32**	**0.11**	**25**	**520**
This work (XNE `TP=128`)	**22nm**	**-**	**0.016**	**0.11**	**112**	**-**
This work (XNE `TP=128`)	**65nm**	**-**	**0.092**	**0.07**	**52**	**-**

TABLE II: Comparison of Hardware Accelerators and Application-Specific ICs for Binary Neural Networks

latter can be run in $2.2\,\mathrm{mJ}$ per frame in real time (8.9 fps). As a third contribution, we also performed an analysis of the relative costs of computation and memory accesses for BNNs, showing how the usage of a hardware accelerator can be significantly empowered by the availability of a hybrid memory scheme.

A prototype based on the MCU system presented in Section IV-B has been taped out in 22nm technology at the beginning of January 2018. Future work includes silicon measurements on the fabricated prototype; the extension of this design to explicitly target more advanced binary neural network approaches, such as ABC-Net [21]; and as more advanced integration with the SRAM memory system to reduce power in high-performance modes and enable more parallel access from the accelerator while keeping the shared memory approach.

REFERENCES

[1] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," *arXiv:1512.00567 [cs]*, Dec. 2015.

[2] Y. Zhang, M. Pezeshki, P. Brakel, S. Zhang, C. L. Y. Bengio, and A. Courville, "Towards End-to-End Speech Recognition with Deep Convolutional Neural Networks," *arXiv:1701.02720 [cs, stat]*, Jan. 2017.

[3] X. W. Chen and X. Lin, "Big Data Deep Learning: Challenges and Perspectives," *IEEE Access*, vol. 2, pp. 514–525, 2014.

[4] M. Dixon, D. Klabjan, and J. H. Bang, "Implementing Deep Neural Networks for Financial Market Prediction on the Intel Xeon Phi," in *Proceedings of the 8th Workshop on High Performance Computational Finance*, ser. WHPCF '15. New York, NY, USA: ACM, 2015, pp. 6:1–6:6.

[5] H. Greenspan, B. van Ginneken, and R. M. Summers, "Guest Editorial Deep Learning in Medical Imaging: Overview and Future Promise of an Exciting New Technique," *IEEE Transactions on Medical Imaging*, vol. 35, no. 5, pp. 1153–1159, May 2016.

[6] A. Loquercio, A. I. Maqueda, C. R. del-Blanco, and D. Scaramuzza, "DroNet: Learning to Fly by Driving," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1088–1095, Apr. 2018.

[7] M. Manic, K. Amarasinghe, J. J. Rodriguez-Andina, and C. Rieger, "Intelligent Buildings of the Future: Cyberaware, Deep Learning Powered, and Human Interacting," *IEEE Industrial Electronics Magazine*, vol. 10, no. 4, pp. 32–49, Dec. 2016.

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

[9] 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," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 9, pp. 2481–2494, Sep. 2017.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv:1512.03385 [cs]*, Dec. 2015.

[11] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," *arXiv:1602.07261 [cs]*, Feb. 2016.

[12] 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," *arXiv:1602.07360 [cs]*, Feb. 2016.

[13] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both Weights and Connections for Efficient Neural Network," in *Advances in Neural Information Processing Systems*, 2015, pp. 1135–1143.

[14] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 243–254.

[15] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients," *arXiv:1606.06160 [cs]*, Jun. 2016.

[16] B. Moons, K. Goetschalckx, N. Van Berckelaer, and M. Verhelst, "Minimum Energy Quantized Neural Networks," *arXiv:1711.00215 [cs]*, Nov. 2017.

[17] A. Pullini, F. Conti, D. Rossi, I. Loi, M. Gautschi, and L. Benini, "A heterogeneous multi-core system-on-chip for energy efficient brain inspired computing," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. PP, pp. 1–1, 2017.

[18] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1," *arXiv:1602.02830 [cs]*, Feb. 2016.

[19] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," in *Computer Vision – ECCV 2016*. Springer, Cham, Oct. 2016, pp. 525–542.

[20] M. Rusci, L. Cavigelli, and L. Benini, "Design Automation for Binarized Neural Networks: A Quantum Leap Opportunity?" *arXiv:1712.01743 [cs, eess]*, Nov. 2017.

[21] X. Lin, C. Zhao, and W. Pan, "Towards Accurate Binary Convolutional Neural Network," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 345–353.

[22] G. Desoli, N. Chawla, T. Boesch, S. p Singh, E. Guidetti, F. D. Ambroggi, T. Majo, P. Zambotti, M. Ayodhyawasi, H. Singh, and N. Aggarwal, "A 2.9TOPS/W deep convolutional neural network SoC in FD-SOI 28nm for intelligent embedded systems," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, Feb. 2017, pp. 238–239.

[23] B. Moons, B. D. Brabandere, L. V. Gool, and M. Verhelst, "Energy-efficient ConvNets through approximate computing," in *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Mar. 2016, pp. 1–8.

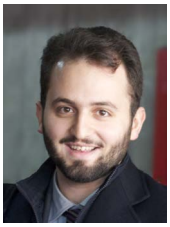
[24] A. Aimar, H. Mostafa, E. Calabrese, A. Rios-Navarro, R. Tapiador-Morales, I.-A. Lungu, M. B. Milde, F. Corradi, A. Linares-Barranco, S.-C. Liu, and T. Delbruck, "NullHop: A Flexible Convolutional Neural Network Accelerator Based on Sparse Representations of Feature Maps," *arXiv:1706.01406 [cs]*, Jun. 2017.

[25] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training Deep Neural Networks with binary weights during propagations," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 3123–3131.

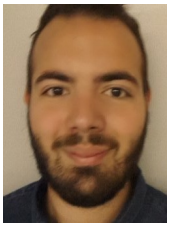
[26] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental Network Quantization: Towards Lossless CNNs with Low-precision Weights," Nov. 2016.

[27] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "YodaNN: An Architecture for Ultra-Low Power Binary-Weight CNN Acceleration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. PP, no. 99, pp. 1–1, 2017.

[28] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. New York, NY, USA: ACM, 2017, pp. 65–74.
[29] 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. PP, no. 99, pp. 1–12, 2017.
[30] L. Jiang, M. Kim, W. Wen, and D. Wang, "XNOR-POP: A processing-in-memory architecture for binary Convolutional Neural Networks in Wide-IO2 DRAMs," in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, Jul. 2017, pp. 1–6.
[31] A. Biswas and A. P. Chandrakasan, "Conv-RAM: An Energy-Efficient SRAM with Embedded Convolution Computation for Low-Power CNN-Based Machine Learning Applications," in *Proceedings of 2018 IEEE International Solid-State Circuits Conference*.
[32] W.-S. Khwa, J.-J. Chen, J.-F. Li, X. Si, E.-Y. Yang, X. Sun, R. Liu, P.-Y. Chen, Q. Li, S. Yu, and M.-F. Chang, "A 65nm 4Kb Algorithm-Dependent Computing-in- Memory SRAM Unit-Macro with 2.3ns and 55.8TOPS/W Fully Parallel Product-Sum Operation for Binary DNN Edge Processors," in *Proceedings of 2018 IEEE International Solid-State Circuits Conference*.
[33] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "UNPU: A 50.6TOPS/W Unified Deep Neural Network Accelerator with 1b-to-16b Fully-Variable Weight Bit-Precision," in *Proceedings of 2018 IEEE International Solid-State Circuits Conference*.
[34] A. A. Bahou, G. Karunaratne, R. Andri, L. Cavigelli, and L. Benini, "XNORBIN: A 95 TOp/s/W Hardware Accelerator for Binary Convolutional Neural Networks," *arXiv:1803.05849 [cs]*, Mar. 2018.
[35] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann, "An Always-On 3.8μJ/86% CIFAR-10 Mixed-Signal Binary CNN Processor with All Memory on Chip in 28nm CMOS," in *Proceedings of 2018 IEEE International Solid-State Circuits Conference*.
[36] F. Conti and L. Benini, "A Ultra-low-energy Convolution Engine for Fast Brain-inspired Vision in Multicore Clusters," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, ser. DATE '15. San Jose, CA, USA: EDA Consortium, 2015, pp. 683–688.
[37] "AMBA 4 AXI4-Stream Protocol Specification."
[38] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2700–2713, Oct. 2017.
[39] A. Pullini, D. Rossi, G. Haugou, and L. Benini, "uDMA: An autonomous I/O subsystem for IoT end-nodes," in *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, Sep. 2017, pp. 1–8.
[40] M. Rusci, D. Rossi, E. Flamand, M. Gottardi, E. Farella, and L. Benini, "Always-ON Visual node with a Hardware-Software Event-Based Binarized Neural Network Inference Engine," in *Proceedings of ACM Computing Frontiers 2018*.
[41] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv:1409.1556 [cs]*, Sep. 2014.
[42] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," *arXiv:1610.02357 [cs]*, Oct. 2016.
[43] "Cypress 64Mbit - 128Mbit HyperRAM Self-Refresh DRAM."



Francesco Conti received the Ph.D. degree from University of Bologna in 2016 and is currently a post-doctoral researcher at the Integrated Systems Laboratory, ETH Zürich, Switzerland and the Energy-Efficient Embedded Systems laboratory, University of Bologna, Italy. He has co-authored more than 20 papers on international conferences and journals. His research focuses on energy-efficient multicore architectures and applications of deep learning to low power digital systems.



Pasquale Davide Schiavone received his B.Sc. (2013) and M.Sc. (2016) in computer engineering from Polytechnic of Turin. Since 2016 he has started his PhD studies at the Integrated Systems Laboratory, ETH Zurich. His research interests include low-power microprocessors design in multi-core systems and deep-learning architectures for energy-efficient systems.

Luca Benini holds the chair of Digital Circuits and Systems at ETH Zürich and is Full Professor at the Università di Bologna. Dr. Benini's research interests are in energy-efficient system design for embedded and high-performance computing. He has published more than 800 papers, five books and several book chapters. He is a Fellow of the ACM and a member of the Academia Europaea. He is the recipient of the 2016 IEEE CAS Mac Van Valkenburg award.