Binary Precision Neural Network Manycore Accelerator

MORTEZA HOSSEINI, University of Maryland, Baltimore County, USA
TINOOSH MOHSENIN, University of Maryland, Baltimore County, USA

This paper presents a low power, programmable, domain-specific manycore accelerator named "BiNMAC"-Binarized neural Network Manycore ACcelerator, which adopts and efficiently executes binary precision weight/activation neural network models. Such networks have compact models in which weights are constrained to only 1 bit and can be packed several in one memory entry that minimizes memory footprint to its finest. Packing weights also facilitates executing single instruction, multiple data (SIMD) with simple circuitry that allows maximizing performance and efficiency. The proposed BiNMAC has light-weight cores that support domain-specific instructions, and a router-based memory access architecture that helps with efficient implementation of layers in binary precision weight/activation neural networks of proper size. With only 3.73% and 1.98% area and average power overhead respectively, novel instructions such as *Combined Population-Count-XNOR*, *Patch-Select* and *Bit-based Accumulation* are added to the instruction set architecture (ISA) of the BiNMAC, each of which replaces execution cycles of frequently used functions with 1 clock cycle that otherwise would have taken 54, 4, and 3 clock cycles respectively. Additionally, customized logic is added to every core to transpose 16×16-bit blocks of memory on a bit-level basis, that expedites reshaping intermediate data to be well-aligned for bit-wise operations. A 64-cluster architecture of the BiNMAC is fully placed and routed in 65 nm TSMC CMOS technology, where a single cluster occupies an area of 0.53 mm^2 with an average power of 232 mW at 1 GHz clock frequency and 1.1 V. The 64-cluster architecture takes 36.5 mm^2 area and, if fully exploited, consumes a total power of 16.4 W, and can perform 1360 GOPS while providing full programmability. To demonstrate its scalability, four binarized case studies including ResNet-20 and LeNet-5 for high-performance image classification, as well as a ConvNet and a multilayer perceptron (MLP) for low power physiological applications were implemented on BiNMAC. The implementation results indicate that the population-count instruction alone can expedite the performance by approximately 5×. When other new instructions are added to a RISC machine with existing population-count instruction, the performance is increased by 58% on average. To compare the performance of the BiNMAC with other commercial-off-the-shelf (COTS) platforms, the case studies with their double-precision floating-point (DPFP) models are also implemented on the NVIDIA Jetson TX2 SoC (CPU+GPU). The results indicate that, within a margin of 2.1% ~ 9.5% accuracy loss, BiNMAC on average outperforms the TX2 GPU by approximately 1.9× (or 7.5× with fabrication technology scaled) in energy consumption for image classification applications. On low power settings and within a margin of 3.7% ~ 5.5% accuracy loss compared to ARM Cortex-A57 CPU implementation, BiNMAC is roughly 9.7× ~ 17.2× (or 38.8× ~ 68.8× with fabrication technology scaled) more energy efficient for physiological applications while meeting the application deadline.

CCS Concepts: • **Computing methodologies → Neural networks**; • **Computer systems organization → Neural networks**; **Heterogeneous (hybrid) systems**; *Reconfigurable computing*; *Real-time system architecture*;

Authors' addresses: Morteza Hosseini, University of Maryland, Baltimore County, 1000 Hilltop Cir. Catonsville, MD, 21250, USA, hs10@umbc.edu; Tinoosh Mohsenin, University of Maryland, Baltimore County, Catonsville, MD, 21250, USA, tinoosh@umbc.edu.

1550-4832/YYYY/0-ARTA $15.00
https://doi.org/0000001.0000001

Additional Key Words and Phrases: BiNMAC, Low power manycore accelerator, Binarized neural network, CPU-GPU, Deep learning, ASIC

ACM Reference Format:
Morteza Hosseini and Tinoosh Mohsenin. YYYY. Binary Precision Neural Network Manycore Accelerator. *ACM J. Emerg. Technol. Comput. Syst.* V, N, Article A (YYYY), 26 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

Artificial neural networks (ANNs) in recent years have attracted tremendous attention in research areas such as voice/image recognition, object localization, keyword spotting, robotics, computer vision, natural language processing, stock market prediction and time-series data processing, to name a few. However, neural networks are commonly computationally intensive and in many cases have large model sizes that require large memory units such as DRAM memory chips that, in embedded systems, are off-chip, power-hungry, and bandwidth-limiting [29]. Additionally, the memory bandwidth to access the neural network parameters on DRAMs restricts the performance and degrades efficiency, which is a considerable factor for wearable and portable devices where the size and energy budgets are severely constricted. Therefore, deploying large neural network models for inference on low-power and resource-bound embedded platforms is challenging.

Deep neural networks (DNNs) for data classification tasks undergo two inter-dependent phases, from design to implementation: 1) training, and 2) testing/inference. The training phase of most of the DNN models relies on parallel computing capability of devices such as multi-core CPUs, graphical processing units (GPUs), and most recently the tensor processing unit (TPU) [21] that can perform high precision float-point operations. For their inference phase, on the other hand, the expensive float-point operations can be altered to low-bitwidth fixed-point operations for which utilizing fixed-point pipelines of general-purpose or application-specific devices can provide a better implementation in terms of performance and energy efficiency.

Many novel efforts were taken in recent years to alter the float-point precision DNN models into compressed models that have less computation complexity and smaller size models, and to provide them for energy-efficient high-performance low-power inference onto custom devices. Some of the most recent of such efforts include parameter quantization [14], network pruning [16], shallow networks [10], neural network model compaction [18], as well as low-bitwidth neural networks such as binary precision weight/activation networks: Courbariaux et al. [8] proposed BinaryConnect (BC), which was the first attempt to train neural networks with weights constrained to either -1 or +1, allowing the weights to be representable by 1 bit (-1 with 0 and +1 with 1), and the multiply-accumulate (MAC) operations to be converted to accumulation only (addition/subtraction). Same authors in [9] proposed Binarized neural networks (BNNs) that constrain not only the weights, but also the activation values to -1 and +1. Rastegari et al. [32] proposed BinaryWeightNetwork (BWN) and XNOR-Net that, similar to BC and BNNs respectively, constrain the weights to 1 bit, but by introducing scale factors that are learned during training, achieve higher accuracy on large datasets such as ImageNet. All of these works have shown that binary precision neural networks can achieve comparable accuracy to full-precision neural networks for datasets such as MNIST, CIFAR-10, and ImageNet.

Nowadays, programmable general-purpose devices that encompass a broad range of consumer electronics including micro-controllers [44], multi-core processors [7], manycore systems [1], as well as edge GPUs [13] are broadly used to implement efficient inference of DNNs. Some of these devices such as the edge TPU [21] from Google include an abundant number of 8-bit multiply-accumulator (MAC) units and dedicate them to DNN models trained as such, thus providing a highly efficient framework for high-performance efficient implementation of DNNs, whereas others such as Xeon processor from Intel [44], on the other hand, incorporate a variety number of instructions and hardware circuitry that allow execution from bitwise operations as requested by BNNs to float-point operations as requested by high-precision DNNs. In addition to general purpose devices, there exists an abundant number of works in the literature [1, 4, 26, 30, 37–40, 45] that propose customized hardware and devices seeking efficiency and performance for implementation of DNNs. The hardware characteristics of a DNN implementation is highly correlated to memory size as well as memory communication and MAC computation overheads that its model impose to a target device. From a data flow and memory communicating point of view, the inference onto a device can be implemented in three approaches [31]: 1) systolic architecture 2) near memory computing (NMC), and 3) in-memory processing (IMP). In systolic array architectures, a DNN can be viewed as a cascade of a few blocks (layers) that perform MAC operations over their input data and generate output data which then constitutes the temporary input data of a next block in the cascade. Thus, by systolically importing/exporting intermediate data through a certain number of hardware MAC units and using a sufficient amount of memory, it is possible to process a DNN from its earliest layer, that is fed by an external input data, to its last layer, that generates the final expected output data. The systolic array designs are typically implemented in general-purpose programming workloads from micro-controllers [44] to manycore systems [1, 11] and most recently to the edge TPU from Google [21]. Aside from the general-purpose devices, there exists an abundant number of other works that propose application-specific systolic architectures to be implementable on FPGAs [30, 37, 38, 45], or to be synthesized using standard cells on ASICs [4, 26, 27, 39, 40]. In near memory computing schemes [6, 22, 23], which can be assumed as the first categories of in memory processing, the objective is to minimize data transfer cost by physically placing compute units in proximity of memory units, to maximize energy efficiency by maximizing the internal bandwidth and minimizing the bottleneck of communication. In Dadiannao [6] as an example, the neural network layers are fed with input data that is fetched directly from nearby buffers. With the advent of emerging fabrication technologies such as resistive random access memories (ReRAMs), unifying computation and communication has become an area of research where memory cells and compute units are designed at a transistor-level to perform load/storage and computation in tandem. Most of the existing single bit ReRAM based accelerator designs focus on binary/ternary precision neural networks [3, 35, 41]. Works like [46] on the other hand, adopt ReRAM technology to implement inference of multi-precision neural networks.

This work is an effort to customize a programmable manycore to accommodate binary precision neural networks for efficient implementation by customizing specific instructions that execute bit-level operations in the proximity of its on-chip distributed memory. In this paper, we propose "BiNMAC"- $\underline{\text{Bi}}$narized neural $\underline{\text{N}}$etwork $\underline{\text{M}}$anycore $\underline{\text{AC}}$celerator, a novel programmable cluster-based manycore, implemented in 65 nm TSMC CMOS technology on a die area of size 36.5 mm^2, with a maximum total power of 16.4 W that is designed to accommodate pre-trained binary precision neural network models for efficient inference. The proposed architecture has novel instructions for different layers of binary precision neural networks, that reduce computations and consequently the inference execution time. We specifically focus on BNN configurations, because it has in common most of the characteristics that other binary precision neural networks including XNOR-Net, BWN, and BC share. Two BNN configurations for image classification have been selected and reconfigured to evaluate the maximum performance of BiNMAC and to compare with an edge GPU implementation, and two other BNN configurations for multi-modal time-series data have been selected to evaluate the energy efficiency and low power characteristics of the BiNMAC compared to low power platforms, such as multi-core CPU. The four case studies are chosen such that for the inference of their binarized models, all the model parameters as well as their intermediate data can

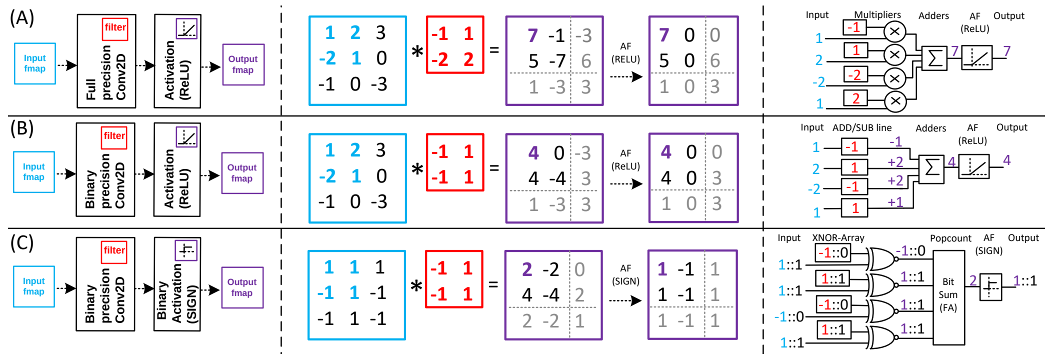


Fig. 1. From left to right, a convolution layer followed by an activation layer, an example of convolution between a 3×3 input and a 2×2 filter going through an activation function, and an equivalent circuitry to infer one output neuron in each example: (A) a full-precision convolution layer, (B) a layer in BC or in the first layer of a BNN, and (C) a layer after the first layer in BNNs. The symbol "::" is used to show the equivalency of -1 and 0.

fit inside the BiNMAC on-chip distributed memory units that constitute a total size of 384KB. The main contributions of this paper include:

- Proposed BiNMAC, a low-power programmable manycore accelerator with special-purpose instructions such as PXNR, PCH, ACCB, and their pertinent special purpose registers, as well as a specific logic that allows transposing memory blocks of 16×16-bit for efficient execution of binarized networks.
- Fully synthesized, and placed-and-routed VLSI layout of 64-cluster BiNMAC in 65 nm TSMC CMOS technology where each cluster contains 3-core processors interfaced with a cluster memory of size 6KB.
- Developed four binarized neural networks for both image processing and physiological applications, including ResNet-20 for CIFAR-10, LeNet-5 for MNIST, a 3-layer MLP for physical activity dataset, and a 7-layer Convolutional neural network (ConvNet) for stress detection dataset that are used for benchmarking and to evaluate BiNMAC in terms of different memory requirements and different number of activated clusters.
- Reconfigured the ResNet-20 for better accuracy and implementation feasibility on BiNMAC after binarization.
- Evaluated the implementation of the four case studies on BiNMAC and compared with relevant work and other COTS products including embedded NVIDIA Jetson TX2 (CPU+GPU).

The rest of this paper is organized into the following sections: Section 2 introduces the full-precision neural networks. Section 3 describes how BNNs work. Section 4 explains the four case studies and our motivations for choosing them. Section 5 discusses the BiNMAC architecture and its evaluation setup. Hardware implementation results for BiNMAC and comparison to GPU and CPU are provided in Section 6. Section 7 provides a comparison of BiNMAC with the related work, and finally, Section 8 presents concluding remarks.

2 FULL-PRECISION NEURAL NETWORKS

Neural networks take in an input data to generate an expected output, and are typically composed of a few layers formed in a directed acyclic graph (DAG) [36], that are made of neurons with trainable weights. Each layer takes in a multi-dimensional array, or *tensor*, performs a dot product over the tensor with trainable weights, and on the result of which applies a non-linear activation function. The generated intermediate data is the *feature map* (fmap) that is then passed to the next layers until the flow of information is propagated to the last layer that generates the expected output, e.g. classification labels. This path is called the *forward pass* and the algorithm with which the trainable weights are deduced from a set of training data is called *back-propagation.* The layers are typically of type *Convolution* (CONV), *Fully Connected* (FC), *Activation Function* (AF), *Batch Normalization* (BN), *Average-Pooling* and *Max-Pooling* (POOL), to name a few. When passing through a layer, depending on the type and the parameters of the layer, the size and dimensions (shape) of the output fmap may deviate from those of the input fmap.

The configuration of a neural network is dataset-specific and is designed empirically, i.e. depending on the size and the type of a dataset, the optimal configuration of the neural network varies. Traditionally, the forward pass is designed by cascading layers, i.e. the output of each layer is connected directly to the input of the next layer. In modern designs, however, it has been shown that for deeper neural networks such as ResNet [17] or DenseNet [19], adding shortcut paths in parallel with the conventional neural network design can help to alleviate the vanishing gradient problem, and hence resulting in higher accuracy for image classification tasks.

Among the aforementioned types of layers, FC and CONV layers contribute to the majority of the implementation bottlenecks including computation time and memory consumption [1, 4]. FC layers are dense with the model weights and can potentially hold most of the network weights. In CONV layers, on the other hand, the weights are packed inside reusable kernels, also known as *filters.* Such layers are used to extract features from the type of data in which there is spatial or temporal correlation within the structure of data, e.g. image (spatial) or voice (temporal). A 2D convolution between a set of 3D filters (that altogether constitute a 4D tensor) $F \in \mathbb{R}^{w_F \times h_F \times d_F \times N_F}$ and a 3D input data $I \in \mathbb{R}^{w_I \times h_I \times d_I}$, results in a 3D output tensor $O \in \mathbb{R}^{w_O \times h_O \times d_O}$ whose elements $O(x, y, z)$ $\forall x, y, z$ are calculated as follows:

$$O(x,y,z) = \sum_{i=0}^{w_F-1} \sum_{j=0}^{h_F-1} \sum_{k=0}^{d_F-1} F(i,j,k,z) I(x+i, y+j, k), \tag{1}$$

where $0 \leq x < w_O$, $0 \leq y < h_O$ and $0 \leq z < d_O$, $w_O/w_F/w_I$, $h_O/h_F/h_I$, and $d_O/d_F/d_I$ are width, height and depth of output/filter/input tensors respectively, and N_F is the number of filters. In 2D convolution with *valid* size, $d_F = d_I$, $w_O = w_I - w_F + 1$, $h_O = h_I - h_F + 1$, and $d_O = N_F$. A convolution between a two-dimensional input and a filter is shown in Fig. 1-A. We suffice to mention that a FC layer with $d_{I_{FC}}$ input nodes and $d_{O_{FC}}$ output nodes is equivalently a 2D CONV layer with $w_d = w_h = w_F = h_F = 1$, $d_I = d_{I_{FC}}$, and $N_F = d_{O_{FC}}$ filters that, if plugged in, convert the equation (1) into a matrix-vector multiplication.

AF is the pivot of neural networks whose non-linearity compared to linear methods provides more degree of freedom for the model to fit a training dataset, thereby higher learnability. Most common non-linear AFs are rectified linear unit (*ReLU*), sigmoid, and tanh functions. A ReLU layer applied to an input data can be mathematically denoted as:

$$O(x,y,z) = \text{ReLU}\left(I(x,y,z)\right) = \max\left(0, I(x,y,z)\right). \tag{2}$$

Other layers such as POOL and Batch Normalization, compared to FC and CONV layers, impose lesser parameters and contribute to negligible computation in a neural network. POOL layer is used to sub-sample the fmap and to convey a more concise representation of fmap information to the next layer. BN [20] suggests that by normalizing and applying a linear transformation to the input of each AF layer, and then another linear shift of the normalized data with two other learnable parameters, the training time can be significantly reduced. It is common practice to use this layer before the AF. Batch Normalization normalizes the distribution of the fmap data using the following equation:

$$O(x, y, z) = \gamma_b \left(\frac{I(x, y, z) - \mu_b}{\sqrt{\delta_b^2 + \epsilon}} \right) + \beta_b, \tag{3}$$

where μ_b and δ_b are the mean and standard deviation over all elements of $I(x, y, z)$ at mini-batch b respectively, ϵ is a pseudo-count to avoid round-off error, γ_b is the trainable scaling and β_b is the offset factor that are calculated per input channel. A Batch Normalization layer has $4 \times d_I$ parameters, that, because of its linear transformation, can be effectively folded into a linear layer (as well as non-linear layer in binarized networks) that it follows or precedes.

3 BINARIZED NEURAL NETWORKS

Similarly, BNNs consist of layers equivalent to those of the full-precision neural networks. However, these layers are implemented differently in BNNs; for the first layer of BNN, data from each input channel (e.g red, green and blue channels for colored images) are in full-precision values, but model weights of the BNN are constrained to either +1 or -1. Therefore, multiply-accumulate operations are replaced by a series of additions and subtractions. From here on, we refer to the first layer of a BNN as a BC layer because it follows rules of BinaryConnect networks [8]. For the next layers, all input features are binarized using binary AFs. The binarized fmap can be represented as a packed binary vector. Therefore, with the packed binarized weights and fmap, a dot product operation between two vectors can be operated by summing (considering 0 as -1) over the result of the bit-wise xnor of the two vectors. The summation over the bits of a register is referred to as $population - count$, and is denoted as pcnt in this paper.

In order to apply the AF to the real-valued variables and transform them into either +1 or -1, deterministic or stochastic binarization functions are proposed by [9]. We use deterministic binarization, since it is more efficient to implement on hardware. The deterministic binarization is essentially a sign function as shown in equation (4),

$$O_b(x, y, z) = \operatorname{sign}\big(I(x, y, z)\big) = \begin{cases} +1 & \text{if } I(x, y, z) \geq 0 \\ -1 & \text{otherwise,} \end{cases} \tag{4}$$

where I is in full-precision and O_b is its binarized tensor.

Since the 2D convolution is the common practice in most of the convolutional neural network architectures including binary precision weight/activation networks, packing binary values along depth (channels) of input and filter tensors–that have the same number of channels–inside memory can facilitate the implementation of equation (1) in hardware. Alternatively, in BNNs a convolution operation can be formulated by equation (5),

$$O(x, y, z) = \sum_{i=0}^{w_F - 1} \sum_{j=0}^{h_F - 1} \operatorname{pcnt}\Big(\operatorname{xnor}\big(F(i, j, :, z), I(x + i, y + j, :)\big)\Big), \tag{5}$$

$F(i, j, :, z) = \{F(i, j, k, z) | k = 0, 1, ..., d_F\}$, which is a vector that has all elements at the $(i + 1)^{th}$ row and the $(j + 1)^{th}$ column of the $(z + 1)^{th}$ filter, and correspondingly $I(i, j, :)$ means a vector that has all elements along depth (channels) of the input tensor at the $(i + 1)^{th}$ row and the $(j + 1)^{th}$ column. Since all of these elements are either 0 or 1, xnor performs bit-wise operation between the two vectors, and pcnt performs a summation over 1s and 0s (with 0 considered as -1) within the xnored result. Fig. 1-A, B and C illustrate three examples of convolution operation between a 3×3 image

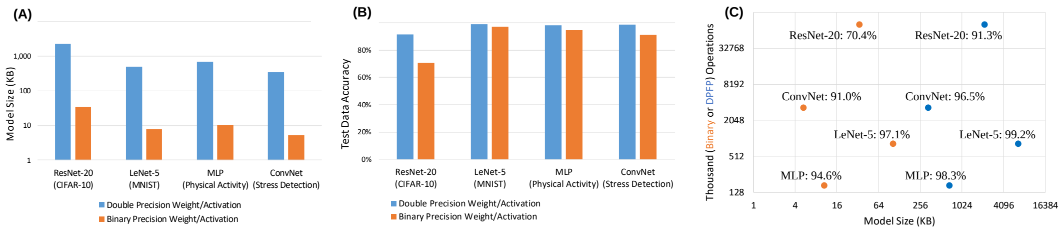


Fig. 2. (A) Required memory, (B) Classification accuracy, and (C) a Memory vs Computation scatter plot for four case studies in this paper with double and binary precision weight/activations. In all of the figures, the colors orange and blue are used for the binary and double-precision floating-point (DPFP) models respectively.

and a 2×2 filter followed by an AF. A hardware schematic which demonstrates the computation for one output node is also depicted for each example, emphasizing the simple circuitry used in BNNs.

The vectorized binary elements along with tensor channels can be packed in memory entries. But in hardware, the memory entries are of fixed size, so the vectors should be chunked and packed in several memory entries, and operated repeatedly per chunked vector until the pcnt-xnor operations of two whole vectors that contribute to the computation of one output value are complete. In a 16-bit machine, if $F(,,:,)$ and $I(,,:)$ in equation (5) are of the size of multiple 16 bits, say $16M$ bits, then the two vectors can be stored in M 16-bit memory entries, and the pcnt-xnor of the two chunked vectors can be performed by accumulating partial sums as in equation (6):

$$\text{pcnt}\Big(\text{xnor}\big(F(,,:,),I(,,:)\big)\Big) = \sum_{m=0}^{M-1} \text{pcnt}\Big(\text{xnor}\big(F(,,16m : 16m+15,),I(,,16m : 16m+15)\big)\Big), \quad (6)$$

where $I(,,16m : 16m+15)$ for $0 \leq m < M$ partitions the $16M$-bit $I(,,:)$ into M chunks of 16-bit entries.

In BNNs, using Batch Normalization is critical and helps not only shortening the training time, but also increasing the accuracy. Utilizing the equation (3) can be further simplified in a BNN that uses a deterministic sign AF layer and, if inserted before the AF, can be merged and represented as an integer bias:

$$sign\Big(\gamma_b\big(\frac{I(x,y,z)-\mu_b}{\sqrt{\delta_b^2+\epsilon}}\big)+\beta_b\Big) = sign\big(I(x,y,z)+\zeta_b\big), \quad (7)$$

where ζ_b for every neuron packs the other 4 parameters of the Batch Normalization before applying the sign AF into a constant bias equal to $(-\mu_b + \frac{\sqrt{\delta_b^2+\epsilon}}{\gamma_b}\beta_b)$ [42]. These 4 parameters are learned during the training phase. Interestingly, due to $I(x,y,z)$ being an integer, ζ_b is not required to be considered in its full precision, but rather the floor of ζ_b instead of real ζ_b given any integer $I(x,y,z)$ still satisfies the equation (7), thus maintaining all the BNN arithmetic in the domain of integer numbers. Nevertheless, the full-precision multiplications imposed by the Batch Normalization layer as in equation (7) are replaced with one fixed-point summation. Therefore, once again a necessity for multiplication is eliminated.

4 CHOICE OF NEURAL NETWORKS AND BINARIZATION

We selected four neural networks as case studies with different characteristics in terms of dataset and configuration, such that their model is storable at on-chip memory units of the BiNMAC. Two

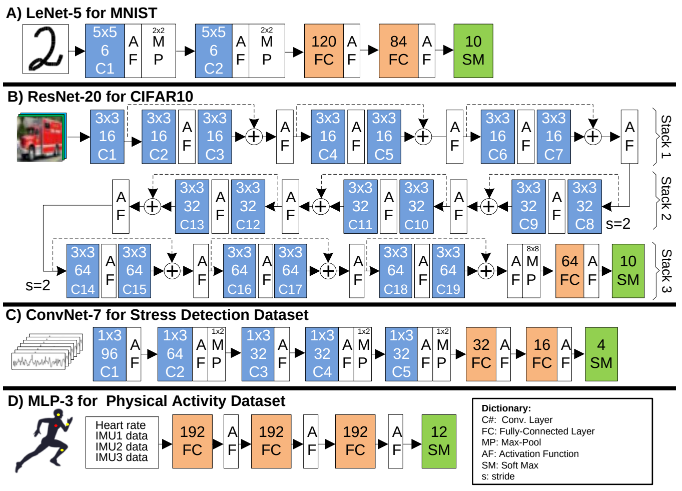


Fig. 3. NN topology of the four case studies in this paper A) LeNet-5 B) ResNet-20 C) ConvNet for Stress Detection D) MLP for Physical Activity Dataset

Table 1. Information for the four different case studies in this paper, their neural network names, model size, and accuracy in both floating-point precision and binary precision, number of operations, and minimum number of required clusters in BiNMAC for implementation of their binarized models. Every multiplication or every addition is considered as one operation.

Network	Applcation	Dataset	Input Shape	# of Labels	CV Params	FC Params	Total Params	BiNMAC Required Clusters (Cores)	CV OPs	FC OPs	Total OPs	FP Acc.	BNN Acc.
LeNet-5	Digit Recognition	MNIST	28×28×1	10	3K	59K	62K	2 (6)	715K	118K	833K	99.2%	97.1%
ResNet-like	Image Classification	CIFAR-10	32×32×3	10	925K	~0K	925K	64 (192)	304M	~0	304M	90.7%	81.8%
MLP	Activity Prediction	PAMAP2	1×1×40	12	0	84K	84K	2 (6)	0	167K	167K	98.3%	94.6%
ConvNet	Stress Detection	EDA	1×64×7	4	9K	33K	42K	1 (3)	3305K	18K	3323K	96.5%	91.0%

of the case studies, LeNet-5 [24] and ResNet-20 [17], are for image classification benchmarks such as CIFAR-10 and MNIST, and are chosen in this work for comparing the performance of the BiNMAC with GPU and the state of the art. The other two, a 3-layer MLP for physical activity dataset [33] and a 7-layer ConvNet for stress detection dataset [5], are for physiological data classification, whose low-power implementation is the main target of the BiNMAC.

In order to evaluate BiNMAC, the four mentioned case studies are initially full-fledged in their full-precision configuration, and then the CONV and FC layers are replaced with equivalent binary layers and train the model from scratch. Binarizing these neural networks causes their double-precision parameter size to shrink by 64× at the expense of losing classification accuracy over their pertinent datasets. This loss is more significant when binarizing the ResNet-20 for CIFAR-10 in which the top-1 validation accuracy drops to 70.4%. In order to compensate for the loss and achieve a similar level of accuracy to the full-precision networks, BNNs parameters may require to be augmented by 2-11× [37]. We note that ResNet-20 has only 273k parameters, whereas the first binarized neural network for CIFAR-10 [9], which has a VGG7-like configuration with an accuracy of 89.9%, has 14,022K parameters. Therefore, in this work, we conducted a set of experiments on a few ResNet-20-like configurations to find one that can fit inside the on-chip memory units (384KB) of the manycore, while giving an acceptable classification accuracy.

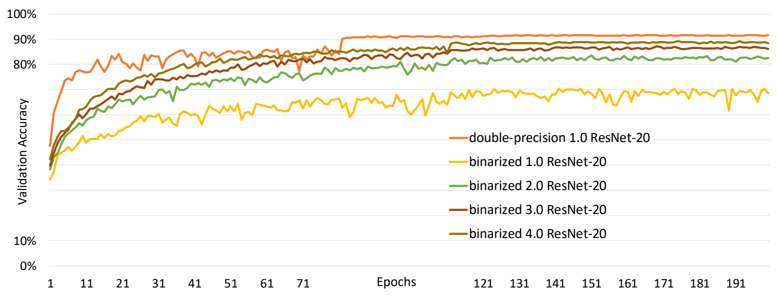


Fig. 4. Training a double-precision ResNet-20 and four BNN ResNet-20 models using augmentation factors. The 1.0 ResNet is the reference model, and the 4.0 ResNet-20 has 4 times as many filter sets as 1.0 ResNet-20 has, thus having 16 times more parameters. By augmenting and binarizing the reference architecture, the accuracy levels of the BNN approaches that of the double-precision model.

Fig. 2-A shows the required memory for the four different neural network architectures, LeNet-5, ResNet-20, as well as the ConvNet and the 3-layer MLP with binary and double-precision weights, and Fig 2-B reflects their classification accuracy on their datasets. Table 1 summarizes the four case studies, their datasets, neural network configurations, and top-1 accuracy when using full-precision versus when converted to a BNN with the original configuration. Within the rest of this section, these four case studies are briefly introduced and described.

4.1 LeNet-5 on MNIST Dataset

MNIST is one of the simplest image classification benchmarking datasets that consists of hand-written digits. The dataset contains 60,000 training and 10,000 test samples. The images are of size 28×28 grey-scaled pixels. Therefore, the input shape is a 2D vector with one channel. LeNet-5 [24] comprises 5 layers: the first 2 of which are CONV layers with filters of size 5×5 and the rest are 2 dense layers followed by a softmax layer. Fig. 3-A shows the neural network topology of LeNet-5 in more detail highlighting the number of filters and parameters in each layer. In total, LeNet-5 for MNIST has 62K parameters and requires 833K operations (Multiplications or Additions) for every image classification. The classification accuracy of LeNet-5 on MNIST is approximately 99.2%. When binarized, the accuracy drops to 97.1%, and the binarized model requires at least 6 cores of the BiNMAC.

4.2 ResNet-20 on CIFAR-10 Dataset

CIFAR-10 is another image classification benchmarking dataset that consists of a training set of 50,000 and a test set of size 10,000 where instances are 32×32 color images that represent 10 objects such as airplanes, automobiles, birds, etc. ResNet [17] for CIFAR-10 is devised by stacking $6n + 2$ weighted layers. We choose $n = 3$ that results in ResNet-20, the smallest model for ResNet, with 3 stacks that comprise 19 CONV layers (plus 6 shortcuts) with filters of size 3×3, and 1 last softmax layer. The three stacks are highlighted in Fig. 3-B and comprise shortcuts that add together fmaps from alternate layers and result in a better accuracy when trained on the full-precision model. ResNet-20 is among the most compact models that can classify CIFAR-10 with an accuracy of 91.3% with 274k parameters. However, when binarizing the ResNet-20, its test accuracy drops to 70.4%. In order to compensate for the accuracy loss, we thicken the channels of the baseline ResNet-20 by increasing the number of filters in each stack per consecutive experiment, thus consequently quadratically increasing the number of parameters and operations per experiment.

Table 2. Effect of binarization and Augmentation of ResNet-20-like configurations

Neural Network Architecture	Stack1 #Filters per Layer	Stack2 #Filters per Layer	Stack3 #Filters per Layer	# of Params	Model Size (KB)	Total Operations	BNN Test Accuracy
1.0 ResNet-20	16	32	64	273k	35	82M	70.4%
2.0 ResNet-20	32	64	128	1077k	136	325M	83.6%
2.0 ResNet-like	**32**	**64**	**128**	**925K**	**116**	**304M**	**81.8%**
3.0 ResNet-20	48	96	192	2,417K	304	736M	87.1%
4.0 ResNet-20	64	128	256	4,308K	540	1,298M	89.2%

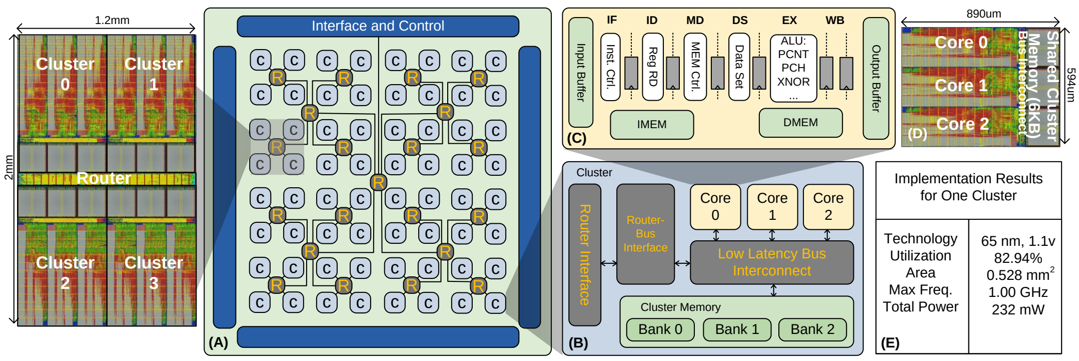
Fig. 4 shows how by thickening the layers of the baseline ResNet-20 through 4 different experiments, the accuracy loss caused by binarization is compensated. We enumerate all these versions via 1.0 ResNet to 4.0 ResNet, where β ResNet-20 is a version whose filters/channels are β times as large as those of the baseline 1.0 ResNet-20. Table 2 shows the result of choosing different number of filters for three other β ResNet-20 configurations. The 2.0 ResNet-like configuration, with an accuracy of 81.8%, is configured by doubling the filters per stack of the 1.0 ResNet-20 and by removing the shortcuts and removing one layer from its third stack, to make it a fit that fully utilizes all resources of BiNMAC. The 4.0 ResNet-20 is made by quadrupling filters of 1.0 ResNet-20. We select the 2.0 ResNet-like from table 2 for evaluating BiNMAC on a BNN with almost all binary CONV layers. From here on, we refer to the 2.0 ResNet-like as the ResNet-like configuration. ResNet-like configuration is further elaborated in table 1 with more details.

4.3 Multi-Layer Perceptron on Physical Activity Dataset

Physical Activity Monitoring (PAMAP2) [33] is a real-world dataset of records of 12 physical activities performed by 9 subjects. The physical activities are labeled as standing, walking, sitting, etc. In order to collect this dataset, one heart rate monitor device and three inertial measurement units (IMUs) were placed on chests, arms, and ankles of the subjects. The sampling frequency of the heart rate monitor device and the IMUs are respectively 9 Hz and 100 Hz. In total, the dataset includes 40 channels of data, where the immediate measurement of all channels given a specific time is indicative of the subject's immediate physical activity. We used the TensorFlow framework and opted out a multilayer perceptron (MLP) from a set of experimental configurations that resulted in an acceptable accuracy on the PAMAP2 dataset. The proposed MLP is a neural network of 3 FC layers whose architecture is depicted in Fig. 3-D. Each layer in the network has a size of 192 neurons, and the ReLU function is used as the AF. The network parameter size, with double precision values, is 667 KB. The network is trained with 90% of the dataset and the classification accuracy over the 10% test dataset over all subjects is 98.3%. When binarized, the accuracy drops to 94.6%, and the binarized model requires at least 6 cores of the BiNMAC.

4.4 ConvNet on Stress Detection Dataset

Additionally, stress detection is included in this work as a physiological application that requires low power implementation settings. The stress detection dataset [5] contains non-EEG physiological signals that are used for inferring the neurological status of a subject. The data was collected from 20 subjects by means of a non-invasive wrist-worn biosensor that collects temperature, 3D acceleration, and the electrodermal activity (EDA) of the subjects. A Nonin 3150 wireless wristOx2 was also used to collect the heart rate. The dataset includes 7 channels, and the status of the subjects is classified as either physical, emotional, cognitive, or relaxation labels. For this dataset, the 7 channel time-series dataset was partitioned into overlapping windows of 1 Sec each containing 64 samples. We then designed a time-series convolutional neural network for an input shape of 1×64×7 that represents the 7 channels, and 64 samples per time frame. The topology of the ConvNet is



Implementation Results for One Cluster	
Technology	65 nm, 1.1v
Utilization	82.94%
Area	0.528 mm^2
Max Freq.	1.00 GHz
Total Power	232 mW

Fig. 5. (A) BiNMAC comprising of 64 clusters (B) Cluster Architecture highlighting cluster memory, low latency bus interconnect and processing cores (C) Processor architecture consisting of 6 pipeline stages where, IF, ID, MD, DS, EX, and WB stages are instruction fetch, instruction decode, memory decode, data set, execution, and write-back stages respectively. IMEM and DMEM are instruction and local data memory respectively. (D) Post-layout view of bus-based cluster implemented in 65nm, 1.1V TSMC CMOS technology (E) Post Layout implementation results of one cluster (consisting of 3 cores + bus + cluster memory)

depicted in Fig. 3-C that comprises 5 CONV layers with filters of size 1×3, followed by 2 dense (FC) layers. The accuracy of the ConvNet in full-precision is 96.5% that dropps to 91% when binarized. The BNN model requires at least 1 cluster (3 cores) of the BiNMAC to be accommodated.

5 BINMAC ARCHITECTURE

5.1 BiNMAC Manycore Overview and Key Features

The BiNMAC accelerator is a stand-alone domain-specific homogeneous manycore platform that is based on Harvard architecture with multiple instruction, multiple data (MIMD) and is designed to accommodate medium-size binarized neural networks and to process their inference such that the budget requirements (power and throughput) are met. The BiNMAC accelerator consists of 64 clusters, with each cluster comprising of a cluster memory of 3072 words (6KB) shared between 3 processing cores with a RISC-like ISA and a 6-stage pipeline, and a low-latency bus for inter-cluster communication between cores and the memory within the cluster, and a hierarchical routing architecture for intra-cluster communication. The processing cores operate on a 16-bit data-path with minimal instruction and data memory (DMEM), making them suitable for task-level and data-level parallelism. These light cores have simplified data memory, instruction memory and ISA to reduce area and power footprint, and to ensure full utilization of their resources when used. Fig. 5-A shows the block diagram of a 64 cluster version of the architecture, highlighting the processing cores in a single bus-based cluster. The clusters are interconnected via a four-ary tree hierarchy of 21 routers that provides message passing between two furthest nodes with 6 router hops.

The components inside the cluster have access to the outer routers via a bus-to-router and router-to-bus interface. Each processing core, cluster bus, cluster memory, and the router was synthesized and fully placed and routed in 65 nm TSMC CMOS technology using Cadence SoC Encounter. Fig. 5-D shows the post-layout view of a bus-based single cluster, where the bus interfaces the three cores to the cluster memory, and provides a link to the outer router. The post-layout implementation results for one cluster are summarized in Fig. 5-E.

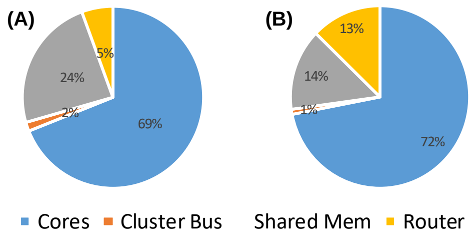


Fig. 6. Post layout implementation breakdown analysis of BiNMAC architecture comprising 192 processing cores, 64 cluster bus, 64 distributed cluster memory units as well as 21 routers. (A) Area breakdown and (B) Power breakdown

5.2 BiNMAC Manycore Platform Evaluation Setup

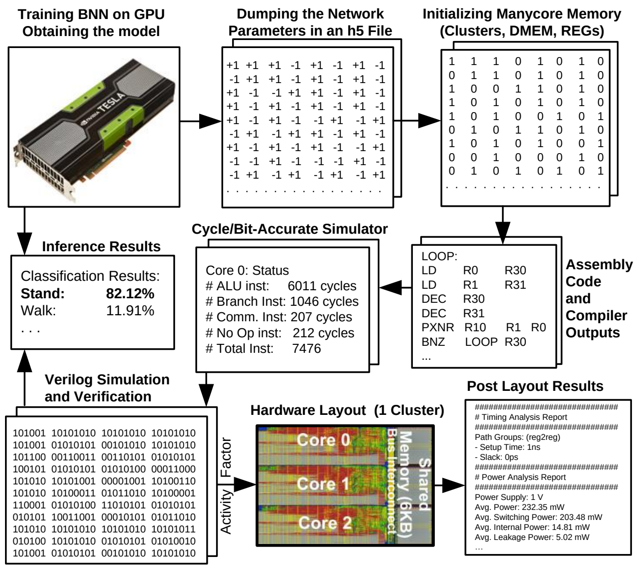
To evaluate the functionality of BiNMAC, we developed a stand-alone simulator and a compiler in Java. This simulator provides cycle/bit accurate results along with task execution and completion time, number of instructions, and memory usage per core. Each application (a BNN classification inference in this work) is first implemented in assembly language for processing cores. The simulator reads in the assembly code as well as an initialized DMEM and cluster memory for each core. It then models the functionality of the processor and calculates the final state of the memory units. The simulator also reports statistics such as the number of clock cycles per core and per application to be used for predicting accurate execution time and throughput. In order to accurately gauge the power and energy consumption analysis of each BNN inference, machine codes obtained from the manycore compiler are mapped on to the hardware description of the manycore platform in Verilog. After initializing the cluster memory units with model weights and the instruction memory (IMEM) units with machine codes, the hardware description model is simulated using Cadence NC-Verilog as shown in Fig. 7. The activity factor is then derived and is used by the Cadence SoC Encounter tool for accurate power estimation for every application. Fig. 6-A shows the area breakdown analysis and Fig. 6-B shows the average power breakdown analysis of the processing cores, cluster bus, cluster memory and router of the 64-cluster BiNMAC architecture. These results are obtained after Place and Route using Cadence Encounter for 65 nm TSMC CMOS technology @1GHZ and 1.1V. The area results are from the post-layout report and the power results are obtained from the Encounter power analysis considering the activity factor, capacitance, and IR drop analysis.

5.3 On-chip Memory Processing

In order to avoid the use of an external DRAM, and to reduce the hardware overhead and extra power consumption, the four binarized case studies were chosen, designed, and reconfigured such that their parameters and temporary fmap can fit inside the on-chip memory of the BiNMAC during the process. By eliminating the need for an external DRAM, not only the hardware overhead is avoided, but the large DRAM memory access delay is eliminated as well, thereby increasing the throughput and lowering the total power consumption. Therefore, BiNMAC, unlike heterogeneous platforms such as TX2, or PENC manycore [28] that require external memory for the neural network model and a CPU for data marshaling, is designed to be standalone.

5.4 Domain Specific Customization of Instruction Set

BiNMAC consists of specialized instructions such as *XNOR*, *PCNT* (population-count), *PCH* (patch-selection), *PXNR* (fused pop-count and xnor), *ACCB* (bit-based accumulation), *BCAST* (broadcast), and *STT* (store transpose of a block) for inference of binary precision neural networks. The former



```
+1 +1  +1  -1  +1  -1  +1  -1
 -1 +1  +1  -1  +1  -1  +1  -1
+1  -1   -1 +1  +1  -1  +1  -1
+1  -1  +1  -1  +1  -1  +1  -1
+1  -1  +1  -1   -1 +1   -1 +1
 -1 +1   -1 +1  +1  -1  +1  -1
+1  -1  +1  -1  +1  -1   -1 +1
 -1  -1  +1  -1   -1  -1  +1  -1
 -1 +1   -1 +1   -1 +1   -1 +1
. . . . . . . . . . . . . . . .
```

```
1  1  1  0  1  0  1  0
0  1  1  0  1  0  1  0
1  0  0  1  1  0  1  0
1  0  1  0  1  0  1  0
1  0  1  0  0  1  0  1
0  1  0  1  1  0  1  0
1  0  1  0  1  0  0  1
0  0  1  0  0  0  1  0
0  1  0  1  0  1  0  1
. . . . . . . . . . . . . . . .
```

```
Classification Results:
Stand:      82.12%
Walk:       11.91%
. . .
```

```
Core 0: Status
# ALU inst:     6011 cycles
# Branch Inst: 1046 cycles
# Comm. Inst: 207 cycles
# No Op inst:   212 cycles
# Total Inst:     7476
```

```
LOOP:
LD      R0      R30
LD      R1      R31
DEC     R30
DEC     R31
PXNR    R10     R1   R0
BNZ     LOOP    R30
...
```

```
101001  10101010  10101010  10101010
101001  01010101  00101010  10101010
101100  00110011  00110101  01010101
100101  01010101  01010100  00011000
101010  10101001  00001001  10100110
101010  10100011  01011010  10100001
110001  01010100  11010101  01010101
010101  10011001  00010101  01011010
101010  10101010  10101010  10101011
010100  10101010  01010101  01010010
101001  01010101  00101010  10101010
```

```
##################################
# Timing Analysis Report
##################################
Path Groups: (reg2reg)
- Setup Time: 1ns
- Slack: 0ps
##################################
# Power Analysis Report
##################################
Power Supply: 1 V
Avg. Power: 232.35 mW
Avg. Switching Power: 203.48 mW
Avg. Internal Power: 14.81 mW
Avg. Leakage Power: 5.02 mW
...
```

Fig. 7. Methodology for a binary precision neural network model to be deployed on the BiNMAC manycore framework: the binarized weights are obtained from training the binarized model on the GPU, and are packed in 16-bit entries. Then, the packed weights are used to initialize memory units along with the assembly program within the Java simulator. The packed weights are meanwhile used along with the machine code for Verilog post-synthesis and post-layout verification. The post-layout report as well as the activity factors are used for power consumption analysis.

five instructions require one clock cycle for the execution stage, and operate on data values stored in either local DMEM or the cluster memory, and the latter two instructions take multiple clock cycles to execute and will be elaborated in following subsections. Adding these domain-specific instructions to the accelerator of binary precision neural networks gives a significant reduction in the number of cycles and lines of programming codes. In addition to the new set of instructions, three special registers, named BACC, PACT, and PACH, are designated to individually accumulate the results of ACCB, PCNT, and PCH instructions respectively. Each of these instructions needs three data inputs read at a time, two of which are the two operands read from the dual-port DMEM and the third is read from the affiliated special purpose accumulators. The programmer can arbitrarily set, reset or move the content of the three special registers to the DMEM.

5.4.1 Bit-based accumulation - ACCB. In order to handle the operations in the first layer of a BNN, or the layers in a BC, where the input data is in full-precision and the weights are binarized, ACCB instruction is devised to perform fused bit-based addition/subtraction. "ACCB R1, R2, B" reads from register R2 and, based on the bit location at index B of R2, adds/subtracts the content of R1 to/from the special purpose accumulator BACC, where R1 and R2 use both direct and indirect addressing modes, and B is an immediate operand. For example, in Fig. 1-B, if the values of the 2×2 filter are aligned in bits [15:12] of R_f (i.e. R_f[15:12] = 4'b0101) and the full-precision elements of the first patch (upper leftmost elements in the patch of 2×2) of the 3×3 image are stored in registers R_{00}, R_{01}, R_{10} and R_{11}, then the four consecutive instructions "ACCB, R_{00}, R_f, 15", "ACCB, R_{01}, R_f, 14", "ACCB, R_{10}, R_f, 13", and "ACCB, R_{11}, R_f, 12" will result in storing number 4 inside the BACC accumulator in 4 execution cycles. In order to avoid data dependency of R_f in consecutive

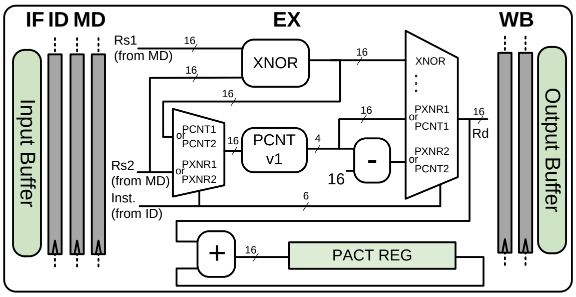


Fig. 8. A compact hardware that integrates components of bit-wise XNOR, PCNT1, PCNT2, PXNR1, and PXNR2. The PXNR is achieved by providing a direct path between the output of the bit-wise XNOR and the input of the PCNT. The special-purpose register PACT accumulates over the result of the PXNR.

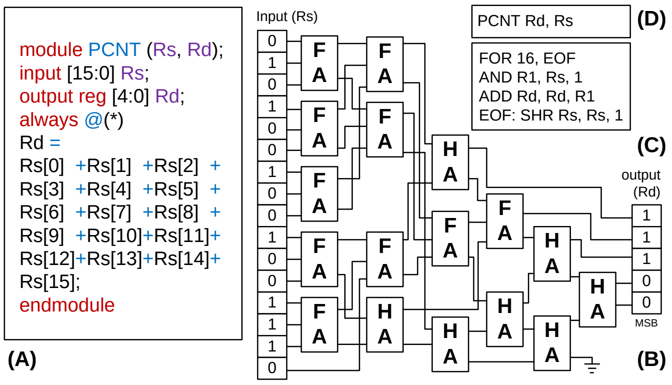
instructions, data forwarding have been handled in the pipeline. ACCB instruction can expedite the convolution between packed binary weights and full-precision input data/fmap by 3× and can manifest its optimization in BCs or in BNNs in which the first layer contribute to the majority of the computation.

5.4.2 XNOR and pop-count instructions. We added XNOR and PCNT and their fused instruction, PXNR, to the ISA of the BiNMAC architecture. XNOR and population-count are the pivot operators of BNNs. Population-count operator counts the number of 1s in a data packet (16 bits in this work) and its functionality can be considered from two points of view: 1) accumulating all 1s in the 16-bit packed data and considering “0" as “0" which is equivalent to a hamming-weight function. 2) considering every “0" as a “-1", and every “1" as a “+1", and then summing over every “+1" and “-1" within a packed data. We call the former and the latter functions population-count versions 1 and 2, and assign PCNT1 and PCNT2 to their instruction opcodes respectively. For example, PCNT1(16’h000F)=4 whereas PCNT2(16’h000F)=-8. For 16-bit input data, both functions are interchangeable with the following equation:

$$\mathrm{PCNT2}(R_s) = 2 \times \mathrm{PCNT1}(R_s) - 16. \tag{8}$$

When synthesizing the two versions individually in hardware, PCNT2 consumes slightly more logic as compared to PCNT1. Therefore, we implement PCNT1 first and by subtracting 16 from whose output infer PCNT2. We note that multiplication by 2 in equation (8) is practically a shifting in wires with no extra logic. Fig. 9 shows the hardware schematic of PCNT1 inside the execution stage, a snippet of Verilog code, the assembly syntax of PCNT1 instruction, and its equivalent assembly function snippet without PCNT1. It is shown that with a very small hardware overhead (10 full adders and 6 half adders) for a data word of size 16, PCNT instruction can replace 4 lines of instruction assembly code, and 52 clock cycles in terms of execution, all fused in one instruction and one clock cycle.

In another attempt to add BNN specialized instructions, the two instructions PCNT and XNOR are fused into one new instruction, named PXNR, by providing a path between the result of the bit-wise XNOR to the input of the PCNT hardware. By so doing, the new instruction generates the result of frequently used combined XNOR and PCNT. “PXNR Rd, Rs1, Rs2" is equivalent to “PCNT Rd2, Rd1" following “XNOR Rd1, Rs1, Rs2", and an extra accumulation of Rd2, replacing 3 execution clock cycles, with one. For PXNR, similar to ACCB, a special purpose register, named PACT, is added that always accumulates the result of PXNR, thereby eliminating the need for further addition in software code, and avoiding one more execution clock cycle. Fig. 8 depicts the schematic that



```
module PCNT (Rs, Rd);
input [15:0] Rs;
output reg [4:0] Rd;
always @(*)
Rd =
Rs[0]  +Rs[1]  +Rs[2]  +
Rs[3]  +Rs[4]  +Rs[5]  +
Rs[6]  +Rs[7]  +Rs[8]  +
Rs[9]  +Rs[10]+Rs[11]+
Rs[12]+Rs[13]+Rs[14]+
Rs[15];
endmodule
```

```
PCNT Rd, Rs
```

```
FOR 16, EOF
AND R1, Rs, 1
ADD Rd, Rd, R1
EOF: SHR Rs, Rs, 1
```

Fig. 9. (A) A Verilog code for PCNT1 (B) An optimal synthesized hardware for PCNT1 instruction composed of 10 Full Adders and 6 Half Adders that generates the population-count from a 16-bit input (C) assembly program snippet that takes 52 clock cycles to emulate population-count (D) equivalent instruction for population-count after incorporating PCNT in the RISC ISA, that fuses 4 lines of machine code and 52 clock cycles into one PCNT1 instruction with 1 clock cycle execution.

integrates the hardware of all the related instructions, XNOR, PCNT1, PCNT2, PXNR1, and PXNR2, as well as the special purpose accumulator PACT, in which resource sharing is used at its finest to minimize the hardware overhead and power consumption.

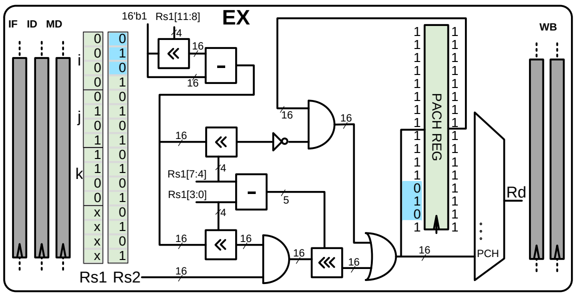


Fig. 10. A compact hardware to implement PCH instruction. Similar to the PACT accumulator, the PACH register is another special register that concatenates in itself patches of arbitrary size from Rs2 appointed by Rs1.

5.4.3 Patch-select - PCH Instruction. Patch-select or PCH is a new invention to the ISA of BiNMAC which facilitates the bit packing and bit manipulation. Since coefficients are packed in 16-bit entries, there are times that a patch of a specific size is required to be read from one data entry and to be shifted and written to another. PCH is devised to meet this necessity and, similar to PCNT, reads a third operand from its special purpose register PACH. For example, "PCH $Rd, Rs2, Rs1$", with $Rs1 = \{i\_j\_k\}$ where i, j, and k are each 4 bits concatenated into Rs1, replaces bit locations $[i+k-1:i]$ (both inclusive) from the PACH register with bit locations $[j+k-1:j]$ from Rs2, and updates both the PACH and the Rd registers in the write-back (WB) stage with the new result. The PCH instruction in BiNMAC has been devised to handle patches of length 1 to 15, and is basically synthesized by four barrel shifters that control the length, start point and end point of a mask. PCH with $i = 1$ is beneficial to packing 16 output bits into one data packet. Fig. 10 shows the schematic hardware of PCH instruction, integrated with the PACH register and their interaction during the execution stage of the BiNMAC.

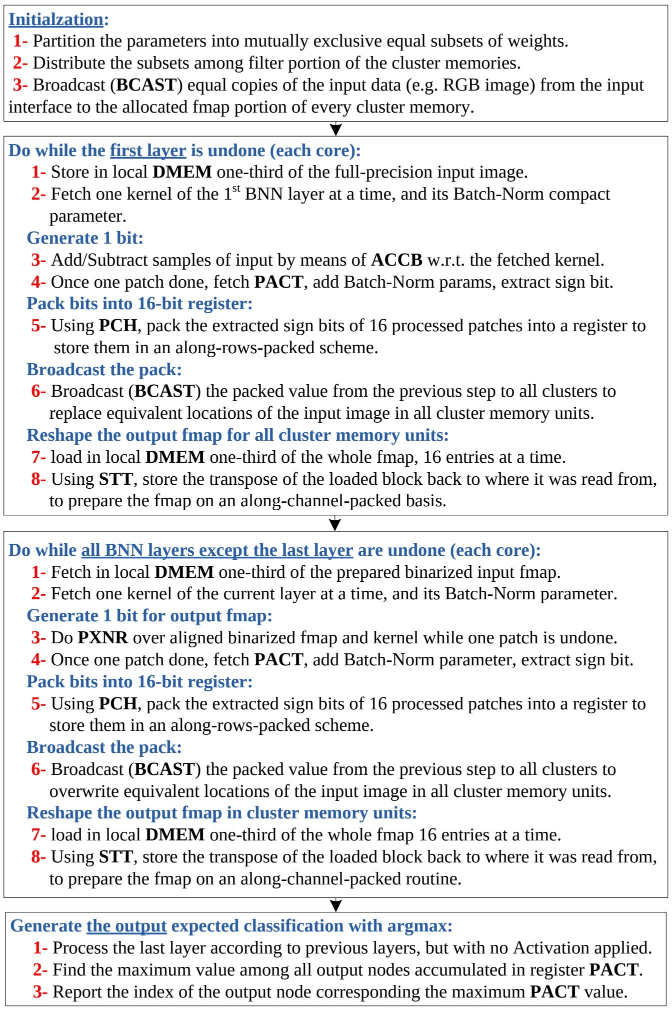
5.4.4 Broadcast - BCAST instruction. In order to accelerate parallel processing when computing a specific BNN layer, we follow the output channel tiling scheme where each cluster requires to have access to the whole input fmap data and process it with the set of filters stored in its cluster memory to generate 1 or a few channels of the output fmap. Thus, the final output fmap is partitioned in every cluster and needs to be totally accessible to all cores in all clusters for the next round of BNN layer processing. BCAST instruction, which is a communication instruction, is added to the ISA of BiNMAC that allows broadcasting data from one core in a cluster to every cluster memory in every corner of the manycore architecture, and is used to provide identical copies of the output fmap for every cluster. In order for the BCAST instruction to function effectively, the original routers were modified to accept a data packet of type broadcast regardless of their source or destination, only to deliver it to an equivalent memory location on every cluster memory. Once such packet is received by a router, it is then passed to every other router, except the sender, that is connected to it until the packet is received by every cluster memory of each cluster in the BiNMAC architecture.

5.4.5 Store transpose of a block - STT instruction. In the next subsection, we will elaborate on the implementation flow of BNNs on BiNMAC, and explain a bottleneck where blocks of memory need to be transposed on a bit-level basis. Thus, it is required to add a mechanism to the hardware that can transpose an arbitrary block of 16×16-bit memory, so to reshape an along-rows-packed channel-major-order stored fmap into an along-channels-packed row-major-order stored fmap inside cluster memory units. Such a function can be naively written with basic RISC instructions including LD/ST, SHR, and bit-wise AND to transpose a block of 16×16-bit memory in 1536 execution cycles. BiNMAC is designed such that for every 16-bit memory read/load, a 16-stage pipeline made of 16 flip flops per stage holds the last 16 loaded memory entries. The 16×16-bit flip flops are wired out such that a transposed pattern of flip flops are multiplexed and writable back to the cluster memory. “STT $addr Fs1" is devised to store 16-bit entry Fs1 from the transposed array of the flip flops back to the address $addr from the cluster memory. Thus, for a block of 16×16-bit memory, it takes 16 LD cycles, and 16 STT cycles (total 32) to be loaded, transposed, and written back to the memory. As an example, “LD $0 R0" through “LD $15 R0" overwrite the register R0 with the first 16 memory entries of the cluster memory, meanwhile the array of 16×16 flip flops gets fully loaded with MEM[0][0:15] through MEM[15][0:15]. Then, “STT $0 F0" writes the already-loaded MEM[0:15][0] back to MEM[0][0:15], and through “STT $15 F15" the transpose of the whole previous 16 memory entries overwrites the first 16 entries of the cluster memory.

In order to compensate for the overhead of adding all the new instructions, we remove the bulky multipliers due to their little usability and extra overhead in BNNs. Multiplication is not used in the case studies of this work. However, if required (e.g. in BWN or XNOR-Net), it should be implemented in a loop by shifting the multiplier operand and accumulating on the shifted multiplicand operand accordingly.

5.5 BNN Implementation Flow

As mentioned in the previous Section for the inference of the BNN, we follow the output channel tiling scheme, in which the computation to communication (CTC) ratio as compared to other tiling schemes is minimum [43], in an intra-cluster level. Thus, for implementation of each case study, all filters in the CONV layers and all the weights in the FC layers are partitioned and distributed among the cluster memory units such that no two memory units from two different clusters share



Initialzation:
1- Partition the parameters into mutually exclusive equal subsets of weights.
2- Distribute the subsets among filter portion of the cluster memories.
3- Broadcast (**BCAST**) equal copies of the input data (e.g. RGB image) from the input interface to the allocated fmap portion of every cluster memory.

Do while the first layer is undone (each core):
1- Store in local **DMEM** one-third of the full-precision input image.
2- Fetch one kernel of the 1st BNN layer at a time, and its Batch-Norm compact parameter.
Generate 1 bit:
3- Add/Subtract samples of input by means of **ACCB** w.r.t. the fetched kernel.
4- Once one patch done, fetch **PACT**, add Batch-Norm params, extract sign bit.
Pack bits into 16-bit register:
5- Using **PCH**, pack the extracted sign bits of 16 processed patches into a register to store them in an along-rows-packed scheme.
Broadcast the pack:
6- Broadcast (**BCAST**) the packed value from the previous step to all clusters to replace equivalent locations of the input image in all cluster memory units.
Reshape the output fmap for all cluster memory units:
7- load in local **DMEM** one-third of the whole fmap, 16 entries at a time.
8- Using **STT**, store the transpose of the loaded block back to where it was read from, to prepare the fmap on an along-channel-packed basis.

Do while all BNN layers except the last layer are undone (each core):
1- Fetch in local **DMEM** one-third of the prepared binarized input fmap.
2- Fetch one kernel of the current layer at a time, and its Batch-Norm parameter.
Generate 1 bit for output fmap:
3- Do **PXNR** over aligned binarized fmap and kernel while one patch is undone.
4- Once one patch done, fetch **PACT**, add Batch-Norm parameter, extract sign bit.
Pack bits into 16-bit register:
5- Using **PCH**, pack the extracted sign bits of 16 processed patches into a register to store them in an along-rows-packed scheme.
Broadcast the pack:
6- Broadcast (**BCAST**) the packed value from the previous step to all clusters to overwrite equivalent locations of the input image in all cluster memory units.
Reshape the output fmap in cluster memory units:
7- load in local **DMEM** one-third of the whole fmap 16 entries at a time.
8- Using **STT**, store the transpose of the loaded block back to where it was read from, to prepare the fmap on an along-channel-packed routine.

Generate the output expected classification with argmax:
1- Process the last layer according to previous layers, but with no Activation applied.
2- Find the maximum value among all output nodes accumulated in register **PACT**.
3- Report the index of the output node corresponding the maximum **PACT** value.

Fig. 11. A general algorithm for a BNN inference using output channel tiling scheme on the BiNMAC

no identical filters in common. Because of the 16-bit architecture of the BiNMAC, filters and input fmap are needed to be packed and chunked along channels and operated according to equations (5) and (6), and the along-channel-packed weights can be stored in memory units on either a column-major-order or a row-major-order (this work) basis. The fmap for every BNN layer is identically replicated in every cluster memory to be locally processed by the 3 cores of the cluster. The filters take up a specific portion of the total on-chip memory space (cluster memory units altogether) and are stored there from the beginning to the end of the inference process. The remaining portion of the cluster memory units is assigned to the intermediate data including the fmap, that is loaded with identical fmap copies in every cluster memory. Thus, the cluster memory should have enough space to accommodate the layer that results in the largest fmap of the neural network model.

Within every cluster, each of the three cores contributes to accomplishing one-third of the process of equation (5) given a filter, using an input tiling scheme in an inter-cluster level: all the three cores load one same filter at a time, and along with loading one-third of partitioned input fmap, generate one-third of the pertinent output channel. For both CONV and FC layers, three tasks of "load" data/filter (LD), "decrement" data/filter pointers (DEC), and "compute" the loaded data/filter (ACCB/PXNR) are frequently repeated in every core within nested loops. The "loading" task brings every core in the cluster to require accessing the shared cluster memory every other 2 clock cycles. Therefore, the choice of three cores per cluster is an ideal designation to get the three cores to access the shared cluster memory one after another and to fully leverage the bandwidth provided by the shared bus. The partitioned input fmaps are slightly overlapped within partitioning borders so the required patches of the input fmap are provided for every processing core.

For an input fmap that has depth (w_d) of multiple of 16 ($16M$), both input fmap and filters are packed and chunked in multiple entries as per equation (6). Thus, in order to generate one output value $O(x, y, z)$ as in equation (5), the PXNR instruction is executed $w_F \times h_f \times M$ times in a nested loop, whose result is accumulated in the special register PACT. If a Batch Normalization layer follows this convolution, the quantized ζ_b is loaded and added to the accumulating PACT register, and if a sign AF follows the Batch Normalization layer, only the sign bit of the final accumulated result is extracted. By using the PCH instruction then, the output fmap bits get packed within the DMEM on an along-rows-packed basis and get ready to be BCAST to their designated addresses allotted to the fmap in every cluster memory. Once broadcast, a previous and obsolete memory location is overwritten with the newly generated packed data. The broadcast packed data from different cores constitute in every cluster memory identical copies of the whole output fmap that are stored along-rows-packed, yet channel-major-order. Finally, with the collaboration of three cores in every cluster, the identical copies are loaded in local data memory units, transposed by means of the STT instruction, and stored back to the local cluster memory to reshape the identical fmap copies into along-channels-packed and row-major-order. At the end of the process of each BNN layer, $N_{clusters}$ replica of fmap, where $N_{clusters}$ is the number of activated clusters, settle in their allotted space within $N_{clusters}$ cluster memory units and are ready to be processed as the next input fmap. The implementation flow is summarized in an algorithm depicted in Fig. 11.

6 IMPLEMENTATION RESULTS AND PLATFORM COMPARISON

In this Section, we evaluate the deployment of each of the four neural networks introduced in Section 4, to either BiNMAC or the NVIDIA TX2's processing components including a 256-core GPU, a quad-core ARM Cortex-A57, and a dual-core Denver CPU, the entire of which fabricated on TSMC 16nm FinFET process. We implement the double-precision floating-point models on the GPU and CPU components, and for the BiNMAC, we implement their binarized counterparts that have approximately 2.1% ∼ 9.5% less accuracy.

Due to the architecture scalability of BiNMAC, for different neural networks, a different number of clusters are activated to accommodate the network parameters and temporary fmaps, and by means of its Dynamic Frequency Scaling (DFS) feature, one can choose the most appropriate frequency given an application. On the one hand, maximizing the number of cores (up to 192 cores) and the frequency (up to 1000MHz) results in the highest performance, but on the other hand, for low power applications, a minimum number of cores and frequency is preferable. In order to minimize the power consumption and meet the required execution time, it is preferred to utilize the least required number of clusters and to reduce the clock frequency to the point that the application deadline is met. One rule of thumb to activate and exploit the adequate number of clusters for a network is to check whether a temporary fmap, as well as all network parameters, fully occupy the minimum number of clusters using the following equation that estimates the least required

Table 3. Hardware implementation results and comparison on BiNMAC and TX2 for two case studies: CIFAR-10 on binarized ResNet-like, and MNIST on binarized LeNet-5. In each column, L and H, based on the activated cores and the frequency setting, represent Low-performance and High-performance configurations. For the TX2, the batch size is set to the maximum amount to fully occupy the GPU memory, and all 4 cores of the ARM CPU and 2 cores of the Denver CPU are activated to provide full data marshaling. The power of the TX2 is reported as the sum of the powers of only the CPU, the GPU and the DDR memory.

Model	**CIFAR-10 for ResNet**						**LeNet-5 for MNIST**					
Platform - Technology	TX2 - 16nm		**BiNMAC - 65nm**				TX2 - 16nm		**BiNMAC - 65nm**			
Configuration	L	H	L	L2L Comparison	H	H2H Comparison	L	H	L	L2L Comparison	H	H2H Comparison
# Active Cores	256	256	192	-	192	-	256	256	6	-	192	-
GPU or BiNMAC Freq.	140	1300	140	-	1000	-	140	1300	140	-	1000	-
CPU Freq. (MHz)	345	2035	-	-	-	-	345	2035	-	-	-	-
Performance (GOPS)	15	72	190	**12.2**×	1360	**18.8**×	13.7	56.5	1.6	0.1×	223.6	**3.9**×
Execution Time (mS)	5.2	1.1	1.6	**3.2**×	0.2	**4.9**×	0.06	0.01	0.52	0.1×	0.004	**3.9**×
Power (W)	3.1	6.5	2.6	**1.2**×	16.4	0.4×	3.2	7.8	0.08	**36.2**×	15.8	0.5×
Classification Energy (mJ)	15.8	7.2	4.1	**3.8**×	3.7	**1.8**×	0.19	0.11	0.04	**4.2**×	0.06	**1.9**×
Projected @ 16nm	-	-	1.0	**15.3**×	0.9	**7.4**×	-	-	0.01	**16.8**×	0.02	**7.6**×
Efficiency (GOPS/W)	5.1	11.1	73	**14.5**×	83	**7.0**×	4.3	7.3	18.1	**4.2**×	14.1	**1.9**×
Projected @ 16nm	-	-	293	**58.0**×	332	**28.0**×	-	-	72.4	**16.8**×	56.4	**7.6**×
Precision	Double (64-bit)		Binary (1-bit)				Double (64-bit)		Binary (1-bit)			
# Parameters	273K		925K				62K		62K			
Model Size (KB)	2184		116				496		8			
# Operations	82M		304M				833K		833K			
Accuracy	91.3%		81.8%				99.2%		97.1%			

number of clusters:

$$N_{clusters} \geq \frac{S_{model}}{C_{cluster} - L_{model}}, \tag{9}$$

where S_{model} is the neural network model size, $C_{cluster}$ is a single cluster memory size which is 6KB, and L_{model} is the largest fmap size within the layers of the given neural network. For example for the binarized ResNet-like, the model has 925K 1-bit parameters that can be packed in S_{model}=116KB. The largest feature map of shape 32×32×32 comes from any layers of the first stack with a packed size of L_{model}=4KB that, when plugged in equation (9), gives $N_{clusters} \geq 57$. Consequently, the number of activated cores is equal to 3×$N_{clusters}$.

The four binarized case studies are implemented and evaluated in two subsections: image recognition case studies for high-performance applications, and multi-modal time-series case studies for low-power application. For every implementation, we use Amdahl's law [2] to evaluate the implementation speedup resulted by the instruction optimization. Amdahl's law states that if a fraction f of a task is accelerated by P times, then the total implementation speedup of that task would be $\frac{1}{1-f+\frac{f}{P}}$ times.

To make a better comparison with different implementation styles, we project our results using technology scaling factors following rule in [15], i.e. constant, linear and quadratic scaling for power, frequency, and area respectively. The method is based upon early findings of Dennard et al. [12] that stopped being correct right around the 65 nm. We acknowledge that such a simplistic method of scaling is just an approximation using equations that generally do not accurately hold in the small modern transistors.

6.1 High-Performance Applications on BiNMAC and GPU

The peak performance of the BiNMAC for BNN layers is 32×192×frequency, which takes place when all the 192 cores perform 1 PXNR instruction, considering every PXNR instruction is equivalent to 16 bit-level multiplication, 15 addition, and 1 accumulation. For BC layers, however, the peak performance is 2×192×frequency where concurrent ACCB instructions perform one MAC operation per core. For the TX2 GPU with 256 cores, this peak is 2×256×frequency, where concurrent floating-point MAC operations are executed in all 256 cores. The image recognition case studies in this work are implemented on BiNMAC using the maximum number of cores and the maximum frequency for each case study, as well as the least number of required cores operating at a low frequency comparable to the minimum frequency of the Pascal GPU in NVIDIA Jetson TX2 SoC. We refer to the two styles as high (H) and low (L) performance implementations and compare them against analogous high and low-performance configurations of the TX2 board. We measure the power consumption of each individual module mounted on the TX2 board using its provided I2C interfacing and, by monitoring the execution time, figure out other implementation characteristics such as energy consumption and performance. Due to the limitation in BiNMAC's on-chip memory, we evaluate the BiNMAC with one testing data instance at a time, which means a batch size of 1. However, since we are seeking the maximum performance in this Section, we set the batch size of the evaluated test data on the TX2 GPU to the maximum amount that GPU memory can accommodate, otherwise, the GPU would be underutilized. Table 3 summarizes the two image classification case studies implemented on low and high-performance settings of TX2 and BiNMAC. The implementation results are in terms of execution time, energy consumption, throughput, performance, and energy efficiency. Each implementation is further elaborated in the next subsection:

6.1.1 Binarized ResNet-like for CIFAR-10. The RGB images of CIFAR-10 are each of size 3KB that, when replicated in each cluster memory, occupy half of the BiNMAC's total memory. The largest fmap of the ResNet-like is of size 4KB and is attributed to every layer of the first stack. As explained in section IV, this BNN was reconfigured such that it fully utilizes all on-chip memory units of the BiNMAC, 33% of which is used to accommodate the network filters and parameters, and the rest to be used for the intermediate data and fmap. Out of 304M operations of the ResNet-like BNN, less than 1% is for the first layer, which is of type BC and is handled by ACCB instruction, and the rest are all BNN layers. Therefore, the ResNet-like BNN is our reference to evaluate the maximum effective performance of the BiNMAC for BNNs. Since every layer of the first stack in the ResNet-like BNN has 32 filters and there are 64 clusters to be used in the BiNMAC, we use a hybrid method of input and output channel tiling as in [29] in order to achieve maximum performance; unlike the layers of the first and second stack in which filter sets can be partitioned and assigned equally to every core, the 32 filters in every layer of the first stack are partitioned in 32 parts and every part of which is replicated identically for two clusters (within the 64-cluster BiNMAC) one of which computes the first half of the output fmap, and the other cluster takes care of the other half. For the layers after the first stack, the slicing and dicing of the computation follows the algorithm of Fig. 11. When the whole flow is run on the 192-core BiNMAC, it takes 223,403 execution cycles for one CIFAR-10 image classification. Compared to the other case studies in this work, the ResNet-like architecture has a significantly large fmap (1-4KB per layer that totals 45KB) that, according to the BNN inference algorithm described in Subsection 5.5, are needed to be transposed layer after layer. This amount of transposition takes 24% of the whole execution time by using the optimized STT instruction. If STT wasn't optimized and the transposing of the fmap would be elongated 48×, according to Amdahl's law [2], the execution time would be deteriorated by 12× (= 76%+48×24%) where 48 is the optimization factor for the STT instruction.

The binarized ResNet-like with accuracy of 81.8% was implemented on BiNMAC using the 64 clusters with 192 cores and with two low (140MHz) and high (1GHz) frequency settings. Also, the original double precision ResNet-20 model with the test accuracy of 91.3% was implemented on the TX2 GPU at two frequency settings (140MHz, and 1.3GHz) to gauge the power, performance, and the classification energy. The results are reflected in Table 3, indicating approximately 19×

more performance and 14× higher efficiency for BiNMAC over TX2 GPU when both set at their maximum settings. Scaling the fabrication technology of the BiNMAC (65nm) to that of the TX2 GPU (16nm), BiNMAC classifies a CIFAR-10 image with 7.4× less energy consumption, however with 9.5% less accuracy. Fig. 12 puts together the performance and power breakdown of all the experienced settings in one plot.

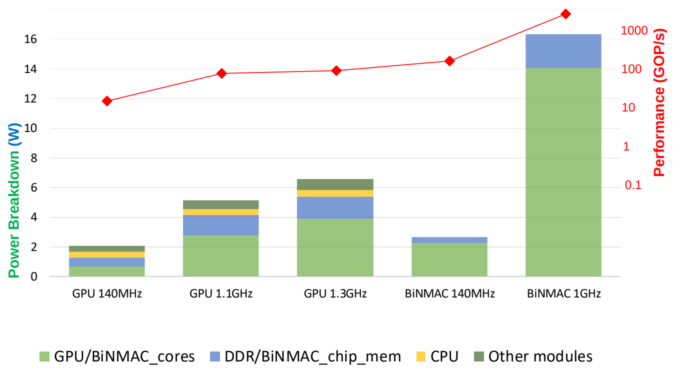


Fig. 12. Power dissipation breakdown of the main components on the TX2 and the BiNMAC with different configuration settings when implementing binarized ResNet-like. The red dot plot shows performance per configuration setting.

6.1.2 Binarized LeNet-5 for MNIST. The binarized LeNet-5 requires 833K operations to classify one single MNIST image, 28% of which is for the first layer, which is of type BC, and on which BiNMAC performs approximately 16× slower than the next layers as a result of the difference in the BiNMAC's peak performance on processing the BC and BNN layers. With much of the computation being carried out in the first layer and with the lower performance of the BiNMAC on BC layer, this layer takes 83% of the total computation time where the ACCB instruction is fetched 27% times of the total fetched instructions during one inference. If ACCB was not optimized and would take 3 clk cycles the total inference performance would be degraded by 1.5× (= 73%+3×27%). Meanwhile, because of the small number of channels in the second layer of the LeNet that results in packing 6 channel bits in every 16-bit register, the PXNR instruction becomes underutilized and performs only 12 operations when used on the along-channel-packed vectors of this layer. When the LeNet-5 is implemented on the BiNMAC and the TX2 with high configuration settings, the performance and efficiency of the BiNMAC are approximately 4× and 2× more than those of the TX2 respectively. With 2.1% less accuracy, the classification energy of one MNIST sample in BiNMAC is 1.9× (7.6× if technology scaled) less than that in the TX2 GPU.

6.2 Low-Power Applications on BiNMAC and CPU

The two BNNs for physiological datasets are implemented on the least required number of cores of the BiNMAC operating at low frequency. The results are compared to the implementation on the ARM Cortex-A57 CPU with 4 activated cores from the TX2 and with the GPU and the dual-core Denver CPU deactivated. In these experiments, we set the batch size of the input test data to 1, because in practice such datum is sampled from low-frequency sensors that can not provide a large batch of data for real-time processing. Meanwhile, when measuring the power consumption on the TX2, we take into account only the power consumption of the CPU. Both the MLP and ConvNet for the physical activity and stress detection datasets are designed such that their model can be

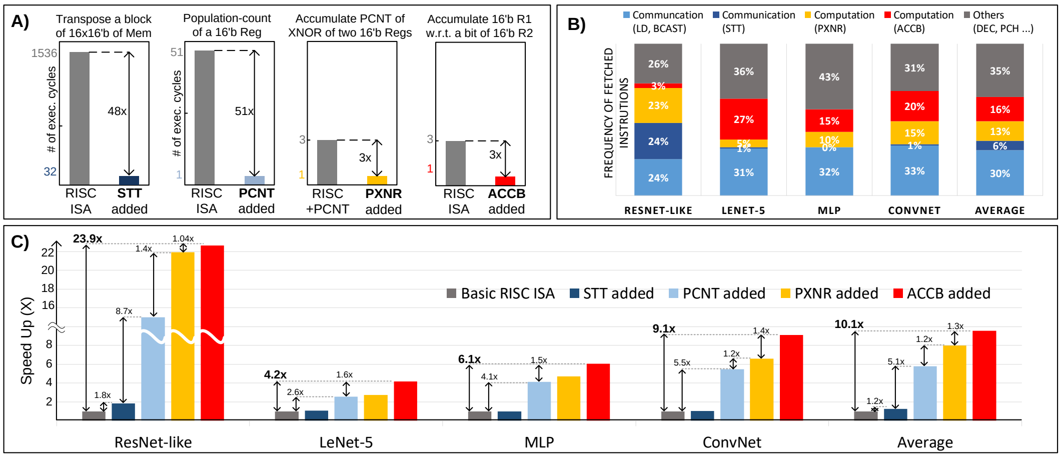


Fig. 13. The impact of adding each of the new instructions to the ISA of the BiNMAC with a RISC instruction set architecture kernel A) the optimization improvement factor for the new instructions as compared to an equivalent function implemented using basic RISC instructions B) the contribution factor of the most fetched instructions on BiNMAC during the implementation of each of the four applications in this paper, C) the speed-up impact of adding the new instructions one after another to the BiNMAC over the implementation performance of each case study.

Table 4. Comparison of low-power biomedical applications implemented on BiNMAC and ARM Cortex-A57 CPU, both configured at low frequency settings and with batch size of 1 input instance

Dataset and Model	**Physical Activity (MLP)**			**Stress Detection (ConvNet)**		
Platform - Technology	CPU - 16nm	**BiNMAC - 65nm**	**Comparison**	CPU - 16nm	**BiNMAC - 65nm**	**Comparison**
Active Cores #	4	6	-	4	3	-
Clock Freq. (MHz)	345	140	-	345	140	-
Performance (GOPS)	0.24	2.67	**11.1×**	0.87	2.05	**2.3×**
Execution Time (mS)	0.70	0.06	**11.1×**	3.79	1.62	**2.3×**
Power (mW)	307	198	**1.5×**	383	92	**4.1×**
Classification Energy (uJ)	214	12.4	**17.2×**	1452	149	**9.7×**
Projected @ 16nm	-	3.1	**68.4×**	-	596	**38.8×**
Efficiency (GOPS/W)	0.8	13.4	**17.2×**	2.3	22.2	**9.7×**
Projected @ 16nm	-	3.4	**68.8×**	-	5.6	**38.8×**
Model Precision	Double (64-bit)	Binary (1-bit)		Double (64-bit)	Binary (1-bit)	
Model Size	672KB	11KB		336KB	6KB	
# Operations	167K	167K		3323K	3323K	
Accuracy	98.3%	94.6%		96.5%	91.0%	

well accommodated in 2 and 1 clusters of the BiNMAC, where 6 cores and 3 cores tile the total computation into 6 and 3 parts respectively. However, similar to the LeNet-5, in the MLP and the ConvNet, the first layers contribute to the majority of computation (44% and 57% respectively.) Overall, for the implementation of the MLP and ConvNet on BiNMAC, the ACCB/PXNR instructions are fetched 15%/10% and 20%/15% of the total fetched instructions, without optimization of which the performance would be degraded 1.5× and 1.7× respectively. Table 4 summarizes the two case studies implemented on the CPU and the BiNMAC. The implementation results are in terms of execution time, energy consumption, energy-delay product, performance, and energy efficiency. On average, the BiNMAC is 13.5 × more energy efficient as compared to the CPU for the implementation of BNNs.

Table 5. Comparison of BiNMAC with the state-of-the-art. Numbers denoted with * are deduced from the information released in the related work

High-Performance Application	[37]	[45]	[30]	[38]	[40]	[4]	[39]	[1]	**This Work**
Implementation Style	FPGA ZC706	FPGA 7Z020	FPGA ZC706	FPGA Zynq7000	ASIC 130nm	ASIC 65nm	ASIC 14nm	ASIC 65nm	ASIC 65nm
Programmability	No	No	No	No	No	No	No	Yes	Yes
Benchmark	CIFAR-10	CIFAR-10	MNIST	MNIST/ SceneLabeling	ImageNet	ImageNet	CIFAR-10/ ImageNet	CIFAR-10	CIFAR-10
Model Precision	binary	binary/ternary	3-bit	INT16	binary	binary	ternary	INT16	binary
Power (W)	11.7	4.7	4.9	-	0.77	0.48	-	4.7*	16.4
Clock Freq (MHz)		143		100	190	400	500	1000	1000
Peak Perf. (GOPS)	-	-	-	-	3501	1510	2500	-	6144
Effective Perf. (GOPS)	2466	208	210	13	876	423	1340	8.7*	1360
Area (mm2)	-	-	-	-	44.9	3.1	2.2	-	36.5
Energy Efficiency (GOPS/W)	211*	44	42	7	1139	879	-	1.9*	82.9
Area Efficiency (GOPS/mm2)	-	-	-	-	19.5	136	600	-	35
Energy Efficiency Projected @65nm	-	-	-	-	2278	879	-	1.9	82.9
Area Efficiency Projected @65nm	-	-	-	-	156	136	5.99	-	35

Fig. 13 encompasses the implementation of the four case studies on the BiNMAC in terms of frequency of fetched instructions and the impact of their optimization. In summary and on average, adding the optimized instructions STT, PCNT, PXNR, and the ACCB to the BiNMAC gradually and one after another speeds up the performance by 1.2×, 5.1×, 1.2×, and 1.3× respectively. The ensemble impact of the latter two instructions accelerates the inference time by 58% and the overall impact of all the optimized instructions is on average 10.1× as compared to the implementation on BiNMAC with only basic RISC instructions.

7 COMPARISON

A large number of works have targeted accelerating deep neural network inference with various model precision on both FPGA and on ASIC [1, 4, 25, 30, 34, 37–40, 45]. Generally, if a well-designed accelerator includes N processing engines (PE), and M MAC units per PE, then it should be able to meet a peak performance of $2 \times N \times M \times Frequency$. Such a performance can be approximately reached in non-programmable devices where the communication bottleneck can be minimized as a result of concurrent implementation to computation logic. In programmable devices, on the other hand, the memory units are communicated using load/store instructions sequential to computation instructions. Thus, such a bottleneck is further highlighted for programmable devices. However, programmablity brings about advantages such as having the device stand-alone and to handle interfacing with different communicating protocols, and to adapt accordingly to the new features emerging from novel neural network configurations. Table 5 compares the BiNMAC implementation results on the ResNet-like architecture with state-of-the-art programmable and non-programmable FPGA/ASIC accelerators for image classification tasks. Fig. 14 shows a scatter plot of performance versus power consumption of the state-of-the-art platforms employed for neural network acceleration. The scatter plot indicates that, in terms of efficiency, BiNMAC manycore stands in a zone between non-programmable and programmable devices.

8 CONCLUSION

We proposed an energy-efficient Binarized neural Network Manycore ACcelerator named "BiNMAC" with specialized and optimized instruction set architecture that incorporates communication and computation instructions, most frequently needed in binary precision weight/activation neural networks, for manipulating data on a bit-level basis. The implementation results indicate that the new instructions can speed up the performance of BNN inference by 1.58× if added to a RISC machine if population-count instruction is already included. Two image recognition case studies were

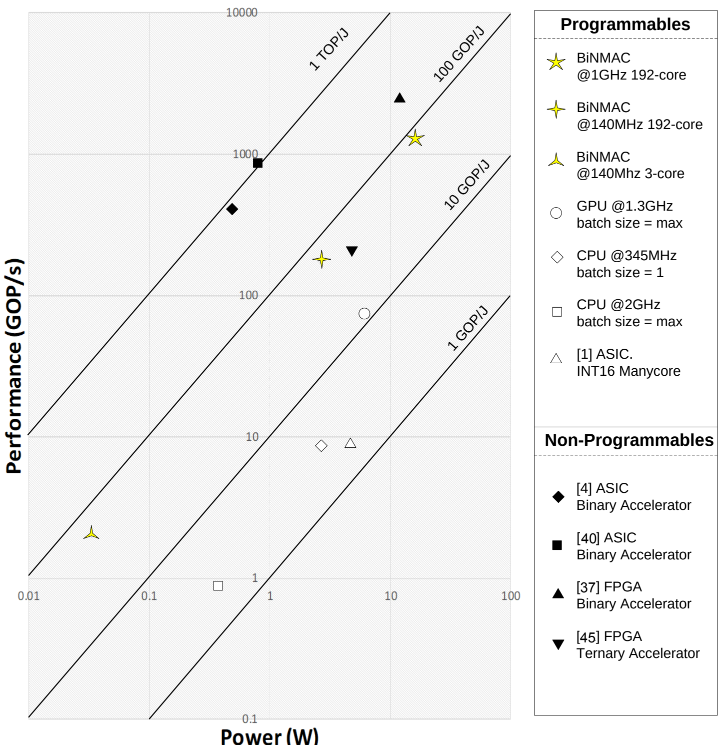


Fig. 14. A performance-power scatter plot to represent the zones where programmable devices such as BiNMAC, CPU, and GPU lie at, versus the zone where non-programmable accelerators stand. Unlike the experiments in Table IV, in this diagram, the ARM CPU is set to perform on the maximum batch size of input data.

implemented on the BiNMAC and on the TX2 SoC both configured with maximum performance settings: when implementing benchmarks such as MNIST with LeNet-5 and CIFAR-10 with ResNet-like BNNs, BiNMAC outperforms the TX2 by 11.3× in performance and by 4.5× in energy efficiency. Two other case studies for physiological datasets including physical activity monitoring and stress detection were implemented on the BiNMAC with minimum power configuration and also on the CPU, to compare the power and efficiency of the two: BiNMAC on average outperforms the ARM Cortex-A57 CPU by 2.8× in power and 13.5× in energy efficiency. Compared to state-of-the-art neural network accelerators, BiNMAC stands in a zone within the power-performance scatter plot between programmable and non-programmable devices.

9 ACKNOWLEDGEMENT

We would like to thank Ali Jafari from Intel for his preliminary work in this paper. We would also like to thank Houman Homayoun from University of California, Davis, and the anonymous reviewers of this work for their valuable inputs and feedback on this project. This research is based upon work supported by the National Science Foundation under Grant No. 1652703.

REFERENCES

[1] Tahmid Abtahi, Amey Kulkarni, and Tinoosh Mohsenin. 2017. Accelerating convolutional neural network with FFT on tiny cores. In *Circuits and Systems (ISCAS), 2017 IEEE International Symposium on*. IEEE, 1–4.

[2] Gene M Amdahl. 1967. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*. 483–485.

[3] Kota Ando, Kodai Ueyoshi, Kentaro Orimo, Haruyoshi Yonekawa, Shimpei Sato, Hiroki Nakahara, Masayuki Ikebe, Tetsuya Asai, Shinya Takamaeda-Yamazaki, Tadahiro Kuroda, et al. 2017. Brein memory: A 13-layer 4.2 k neuron/0.8 m synapse binary/ternary reconfigurable in-memory deep neural network accelerator in 65 nm cmos. In *VLSI Circuits, 2017 Symposium on*. IEEE, C24–C25.

[4] Renzo Andri, Lukas Cavigelli, Davide Rossi, and Luca Benini. 2017. YodaNN: An architecture for ultralow power binary-weight CNN acceleration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 1 (2017), 48–60.

[5] Javad Birjandtalab, Diana Cogan, Maziyar Baran Pouyan, and Mehrdad Nourani. 2016. A Non-EEG Biosignals Dataset for Assessment and Visualization of Neurological Status. In *Signal Processing Systems (SiPS), 2016 IEEE International Workshop on*. IEEE, 110–114.

[6] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. 2014. Dadiannao: A machine-learning supercomputer. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 609–622.

[7] Philip Colangelo, Randy Huang, Enno Luebbers, Martin Margala, and Kevin Nealis. 2017. Fine-grained acceleration of binary neural networks using intel® xeon® processor with integrated fpga. In *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 135–135.

[8] Matthieu Courbariaux et al. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*. 3123–3131.

[9] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830* (2016).

[10] Yann N Dauphin and Yoshua Bengio. 2013. Big neural networks waste capacity. *arXiv preprint arXiv:1301.3583* (2013).

[11] Lei Deng, Zhe Zou, Xin Ma, Ling Liang, Guanrui Wang, Xing Hu, Liu Liu, Jing Pei, Guoqi Li, and Yuan Xie. 2018. Fast object tracking on a many-core neural network chip. *Frontiers in neuroscience* 12 (2018), 841.

[12] Robert H Dennard, Fritz H Gaensslen, Hwa-Nien Yu, V Leo Rideout, Ernest Bassous, and Andre R LeBlanc. 1974. Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Journal of Solid-State Circuits* 9, 5 (1974), 256–268.

[13] Dustin Franklin. 2017. Nvidia jetson tx2 delivers twice the intelligence to the edge. *NVIDIA Accelerated Computing| Parallel Forall* (2017).

[14] Yunchao Gong et al. 2014. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115* (2014).

[15] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 243–254.

[16] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[18] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[19] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4700–4708.

[20] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[21] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*. 1–12.

[22] Phil Knag, Gregory K Chen, Raghavan Kumar, Huseyin Ekin Sumbul, and Ram Krishnamurthy. 2020. Energy efficient compute near memory binary neural network circuits. US Patent App. 16/697,616.

[23] Youngeun Kwon, Yunjae Lee, and Minsoo Rhu. 2019. TensorDIMM: A Practical Near-Memory Processing Architecture for Embeddings and Tensor Operations in Deep Learning. In *Proceedings of the 52nd Annual IEEE/ACM International*

Symposium on Microarchitecture. 740–753.

[24] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[25] Ali Mirzaeian, Houman Homayoun, and Avesta Sasan. 2019. Tcd-npe: A re-configurable and efficient neural processing engine, powered by novel temporal-carry-deferring macs. In *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 1–8.

[26] Ali Mirzaeian, Houman Homayoun, and Avesta Sasan. 2020. NESTA: Hamming Weight Compression-Based Neural Proc. EngineAli Mirzaeian. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 530–537.

[27] Eriko Nurvitadhi et al. 2016. Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC. In *Field-Programmable Technology (FPT), 2016 International Conference on*. IEEE.

[28] Adam Page, Nasrin Attaran, Colin Shea, Houman Homayoun, and Tinoosh Mohsenin. 2016. Low-power manycore accelerator for personalized biomedical applications. In *Proceedings of the 26th edition on Great Lakes Symposium on VLSI*. ACM, 63–68.

[29] Adam Page, Ali Jafari, Colin Shea, and Tinoosh Mohsenin. 2017. SPARCNet: A Hardware Accelerator for Efficient Deployment of Sparse Convolutional Networks. *J. Emerg. Technol. Comput. Syst.* 13, 3, Article 31 (May 2017), 32 pages. https://doi.org/10.1145/3005448

[30] Jinhwan Park and Wonyong Sung. 2016. FPGA based implementation of deep neural networks using on-chip memory only. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE, 1011–1015.

[31] Xiaochen Peng, Minkyu Kim, Xiaoyu Sun, Shihui Yin, Titash Rakshit, Ryan M Hatcher, Jorge A Kittl, Jae-sun Seo, and Shimeng Yu. 2019. Inference engine benchmarking across technological platforms from CMOS to RRAM. In *Proceedings of the International Symposium on Memory Systems*. 471–479.

[32] Mohammad Rastegari et al. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*. Springer, 525–542.

[33] Attila Reiss and Didier Stricker. 2012. Introducing a new benchmarked dataset for activity monitoring. In *Wearable Computers (ISWC), 2012 16th International Symposium on*. IEEE, 108–109.

[34] Colin Shea and Tinoosh Mohsenin. 2019. Heterogeneous scheduling of deep neural networks for low-power real-time designs. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 15, 4 (2019), 1–31.

[35] Linghao Song, You Wu, Xuehai Qian, Hai Li, and Yiran Chen. 2019. ReBNN: in-situ acceleration of binarized neural networks in ReRAM using complementary resistive cell. *CCF Transactions on High Performance Computing* 1, 3 (2019), 196–208.

[36] Krishnaiyan Thulasiraman and Madisetti NS Swamy. 1992. *Graphs: theory and algorithms*. Wiley Online Library.

[37] Yaman Umuroglu et al. 2017. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the SIGDA*. ACM.

[38] Stylianos I Venieris and Christos-Savvas Bouganis. 2016. fpgaConvNet: A framework for mapping convolutional neural networks on FPGAs. In *Field-Programmable Custom Computing Machines (FCCM), 2016 IEEE 24th Annual International Symposium on*. IEEE, 40–47.

[39] Ganesh Venkatesh, Eriko Nurvitadhi, and Debbie Marr. 2017. Accelerating deep convolutional networks using low-precision and sparsity. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*. IEEE, 2861–2865.

[40] Yizhi Wang, Jun Lin, and Zhongfeng Wang. 2017. An Energy-Efficient Architecture for Binary Weight Convolutional Neural Networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2017).

[41] Shihui Yin, Zhewei Jiang, Jae-Sun Seo, and Mingoo Seok. 2020. XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks. *IEEE Journal of Solid-State Circuits* (2020).

[42] Haruyoshi Yonekawa and Hiroki Nakahara. 2017. On-Chip Memory Based Binarized Convolutional Deep Neural Network Applying Batch Normalization Free Technique on an FPGA. In *Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017 IEEE International*. IEEE, 98–105.

[43] Chen Zhang et al. 2015. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 161–170.

[44] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. 2017. Hello edge: Keyword spotting on microcontrollers. *arXiv preprint arXiv:1711.07128* (2017).

[45] Ritchie Zhao et al. 2017. Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs.. In *FPGA*. 15–24.

[46] Zhenhua Zhu, Hanbo Sun, Yujun Lin, Guohao Dai, Lixue Xia, Song Han, Yu Wang, and Huazhong Yang. 2019. A Configurable Multi-Precision CNN Computing Framework Based on Single Bit RRAM. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.