OrthrusPE: Runtime Reconfigurable Processing Elements for Binary Neural Networks

Nael Fasfous[1*], Manoj Rohit Vemparala[2*], Alexander Frickenstein[2*], Walter Stechele[1]
[1]*Department of Electrical and Computer Engineering, Technical University of Munich, Munich, Germany*
[2]*Autonomous Driving, BMW Group, Munich, Germany*
[1]{nael.fasfous, walter.stechele}@tum.de, [2]{manoj-rohit.vemparala, alexander.frickenstein}@bmw.de

***Abstract*—Recent advancements in Binary Neural Networks (BNNs) have yielded promising results, bringing them a step closer to their full-precision counterparts in terms of prediction accuracy. These advancements were brought about by additional arithmetic and binary operations, in the form of scale and shift operations (fixed-point) and convolutions with multiple weight and activation bases (binary). In this paper, we propose OrthrusPE, a runtime reconfigurable processing element (PE) which is capable of executing all the operations required by modern BNNs while improving resource utilization and power efficiency. More precisely, we exploit DSP48 blocks on off-the-shelf FPGAs to compute binary Hadamard products (for binary convolutions) and fixed-point arithmetic (for scaling, shifting, batch norm, and non-binary layers), thereby utilizing the same hardware resource for two distinct, critical modes of operation. Our experiments show that common PE implementations increase dynamic power consumption by 67%, while requiring 39% more lookup tables, when compared to an OrthrusPE implementation.**

I. Introduction

Convolutional Neural Networks (CNNs) have become the state-of-the-art in image classification, semantic segmentation and other computer vision tasks [1]–[3]. Depending on the complexity of the task and the network, CNNs can range from being relatively lightweight to having extreme compute and memory requirements. Furthermore, many applications which require on-edge inference set a tight constraint on power and latency. These challenges gave rise to diverse efforts in optimizing neural networks, which can coarsely be categorized as structural, algorithmic and hardware optimizations.

Algorithmic optimizations concern the mathematical method of executing the convolution operation [4]. Hardware optimizations are related to the dataflow and reuse strategies, as well as the overall architecture on which the network is executed. Structural optimization lays emphasis on the composite structure of the network. Popular methods for structural optimization include quantization, pruning, and filter decomposition among others [5], [6].

Naturally, such optimizations are often unfruitful in cases where the underlying hardware cannot exploit the improvements. This led to hardware development aimed at capitalizing on the aforementioned optimization methods [7].

Quantization is one of the direct methods capable of reducing a neural network's memory footprint, and potentially simplifying the compute units tasked with carrying out the Multiply-Accumulate (MAC) operations, which make up the bulk of the network. Floating-point weights and activations have been proven to be superfluous for inference in most tasks, making fixed-point representation an attractive alternative. However, as the number of bits for the fixed-point representation decreases, accuracy degradation ramps up and more of the critical, learned information is lost. Nonetheless, a particular trend takes this form of optimization to its extremity, fully binarizing the network and creating Binary Neural Networks (BNNs).

*Corresponding Authors

A. Challenges for Binary Neural Networks

BNNs have been researched in previous works [8]–[12]. Different forms of implementation for BNNs have been explored, namely binary weights or binary activations, as well as binary weights *and* activations. Intuitively, having both weights and activations in the binary format represents the highest loss of information, albeit while providing the most memory, energy and compute efficient solution. The billions of multiplications typically required for forward passes are reduced to simple XNOR logic operations, while the accumulations are translated to population counts (popcounts) [12].

The main issue with such a low information representation is that the cumulative effect of the finely-tuned, learned weights of the network is lost. Early attempts at realizing fully binarized convolutional neural networks resulted in accuracies far below those achieved by state-of-the-art CNNs on complex problems such as ImageNet [8].

To counter this problem, Lin et al. [11] approximated activations and weights using binary bases to create Accurate Binary Convolutional Networks (ABC-Nets). Using this method, BNNs have achieved accuracies only 4-5% below their full-precision counterparts for Top-1 and Top-5 results on ImageNet. This is elaborated in section III.

B. Motivation for Reconfigurable BNN Processing Elements

Although binary bases present a solution to the information loss problem, they also introduce new operations to be executed on hardware. Furthermore, the first layer of the neural network is critical and it maintains fixed-point values for weights and activations to avoid severe loss of information at the input of the network. Lastly, we cannot understate the effect of batch normalization on improving the accuracy and training time of BNNs [9].

In this work, we provide a hardware solution applicable to virtually all[1] Xilinx FPGAs for accelerating accurate binary neural networks, without employing additional hardware for the non-binary operations that need to be performed at intermediate stages. We show that an off-the-shelf FPGA with DSP48 blocks can be reconfigured at runtime to compute the highly parallel binary Hadamard products required for binary convolution operations, as well as the fixed-point operations that occur intermediately. We compare synthesis and implementation results of processing elements (PEs) using our method, OrthrusPE[2], against other configurations with equivalent throughput. Our method is orthogonal to dataflow optimizations and the overall accelerator architecture, since it can be implemented as an extension to any existing, compliant FPGA accelerator.

The contributions of this work are summarized as follows:

- A flexible computation unit for accelerating a wide range of BNNs (e.g. Tab. I).
- Execution of SIMD-based binary Hadamard product on FPGA hard blocks.
- A novel, runtime reconfigurable processing element which dynamically supports binary and fixed-point computations.
- Formalization of the relationship between computation mode switching and partial result memory for BNN layers with multiple binary bases.

II. Related Work

Several accelerators have been developed with processing elements designed to exploit performance boosts due to variable quantization levels [13]–[16]. Other accelerators were designed to solely execute BNNs [17]–[19].

Bit Fusion [13], UNPU [14], Stripes [15] and Loom [16] are all based on ASIC designs. UNPU, Stripes and Loom offer single bit operations while the Bitbricks structure used in Bit Fusion allow for the execution of operations at fixed quantization levels, making the smallest possible precision bounded by the size of a single Bitbrick. UNPU, Stripes and Loom are capable of performing both binary and fixed-point operations, however, with a non-negligible overhead, due to the support of variable quantization levels. Further ASIC-based works, BRein [18] and YodaNN [17], were developed precisely to accelerate BNNs. However, they do not implement the binary bases required for accurate binary nets, nor do they support the shifting and scaling of the intermediate maps.

FINN [19] is a popular framework for accelerating BNNs on FPGAs. The framework is geared towards BNNs similar to the ones proposed in [8]. FINN compiles HLS code from a BNN description to create a bit file that exactly suits that network. The authors state that they do not binarize the first and last layers in their *CNV* network, which leads one to conclude that the hardware being utilized for those layers is instantiated for the sole purpose of executing only those two parts of the network. In our proposed solution, every instantiated DSP can be used for any part of the entire network, due to the dual modes of the PE that can be reconfigured at runtime. Furthermore, FINN is not compatible with multiple binary bases, but rather simpler BNNs suited for problems such as MNIST, CIFAR-10 or SVHN. Other FPGA-based BNN accelerators [20]–[22] also execute binary operations purely on LUTs and utilize DSPs for fixed-point operations, where they are supported.

The authors of Double MAC [23] extract more functionality from FPGA hard blocks. They precondition the signals going into DSP blocks such that two multiplications can be obtained with some post-processing. This leverages quantization, since the two results obtained at the output are calculated from operands that are smaller than the maximum possible precision that the DSP can offer. Their work virtually turns DSPs into SIMD multipliers. Similarly, our proposed solution turns DSPs into SIMD binary Hadamard product processing units. Our solution is orthogonal to Double MAC, making it possible to include Double MAC as a *third* mode in OrthrusPE.

Efficient exploitation of hard blocks on FPGAs can play a key role in lowering the efficiency gap between ASIC and FPGA implementations [24]. This is evident in the recent trend of FPGA manufacturers adding more hard blocks to their chips aimed at accelerating deep neural network applications [25].

III. Binary Neural Networks

This section outlines the basic building blocks of BNNs in order to identify the hardware functions required by OrthrusPE. Different components of the BNNs proposed in literature are summarized in Tab. I. In BNNs, the weights and activation of a neural network are constrained to $\{-1, +1\}$. In the hardware implementation, the '-1' values are mapped to '0', in order to execute logical operations (i.e. popcount or XNOR) during inference. Throughout this section, this affine transformation is considered and we define $\mathbb{B} = \{0, 1\}$.

As discussed in Section I-A, modern BNNs use multiple weight and activation bases to reduce the gap in prediction accuracy between the binarized network and its full-precision counterpart. As opposed to previous implementations, where the values were binarized using a $sign()$ function, Lin et al. [11] presented a solution where the values are scaled, shifted and then binarized. This produces multiple unique binary bases which preserve more information collectively, due to the additional operations performed before obtaining them.

Without loss of generality, an activation $A^{l-1} \in \mathbb{R}^{X_i \times Y_i \times C_i}$ is considered as an input to a convolutional layer $l \in [1, L]$ of an L-layer CNN, where X_i and Y_i indicate the input spatial dimensions, and C_i is the number of input channels. Moreover, the weights $W^l \in \mathbb{R}^{K \times K \times C_i \times C_o}$ are the trainable parameters of the layer, where K and C_o are the kernel dimensions and output channels respectively. Throughout training, the trainable parameters converge to either ± 1.

[1]All 7-series, Ultrascale and Ultrascale+ FPGAs, as well as all ZYNC SoCs (DSP48E1 and DSP48E2). The entry level Spartan-6 presents the only exception (DSP48A1).

[2]In accordance to the Greek mythographer Apollodorus, Orthrus is a two-headed dog.

TABLE I
REQUIREMENTS OF MOST COMMON BINARY NEURAL NETWORKS AND THE RESPECTIVE HARDWARE OPERATIONS FOR EXECUTION.

Method	**Binary Weights \Activations**	**Weight Scale α**	**Activation Scale β**	**Batch-Norm**	**Multiple Weight Bases M**	**Multiple Activation Bases N**
BNN [8]	✓	✗	✗	✓	✗	✗
XNOR-Net [12]	✓	✓	✓	✓	✗	✗
CompactBNN [26]	✓	✗	✓	✓	✗	✓
ABC-Net [11]	✓	✓	✓	✓	✓	✓
Hardware Operation	XNOR-Popcount	Multiplication	Multiplication	Multiplication-Shift	MAC	MAC

The sign, scale and shift functions are used to find an appropriate binarization for A^{l-1} and approximate it into $H^{l-1} \in \mathbb{B}^{X_i \times Y_i \times C_i \times N}$, having N binary bases. In a similar manner, W^l is approximated as $B^l \in \mathbb{B}^{K \times K \times C_i \times C_o \times M}$, where M is the number of weight bases. Eq. 1 represents the binary convolutional layer.

$$A^l = \texttt{Conv}(B^l, H^{l-1}) \tag{1}$$

Eq. 2 demonstrates the binary convolution using sub-tensors $B^l_m \in \mathbb{B}^{K \times K \times C_i \times C_o}$ and $H^{l-1}_n \in \mathbb{B}^{X_i \times Y_i \times C_i}$.

$$A^l = \sum_{m=1}^{M} \sum_{n=1}^{N} \alpha_m \beta_n \texttt{BinConv}(B^l_m, H^{l-1}_n) \tag{2}$$

Consider a binary weight tensor slice $b^l \subset B^l_m$ where $b^l \in \mathbb{B}^{K \times K}$. The activation slice $h^{l-1} \subset H^{l-1}_n$ where $h^{l-1} \in \mathbb{B}^{X_i \times Y_i}$ is defined accordingly. Eq. 3 shows the XNOR operations performed on b^l and h^{l-1}, which is the fundamental operation of the BNN. The XNOR operations are grouped as binary Hadamard products of the sliding kernel windows. Next, the partial sum p_{m,n,c_i} is the accumulation of the intermediate XNOR operations over the kernel dimensions K.

$$\begin{aligned} \underbrace{p_{m,n,c_i}}_{\text{fixed-point}} &= \texttt{popcnt}(\texttt{xnor}(b_{k_x,k_y}, h_{x_i+k_x,y_i+k_y})) \\ &= \sum_{k_x=1}^{K} \sum_{k_y=1}^{K} \texttt{xnor}\,\underbrace{(b_{k_x,k_y}, h_{x_i+k_x,y_i+k_y})}_{\text{binary}} \end{aligned} \tag{3}$$

Finally, to compute a single output pixel $a_{m,n}$ relative to bases m and n, the popcount values p_{m,n,c_i} need to be accumulated across the input channels C_i , as shown in Eq. 4.

$$a_{m,n} = \sum_{c_i=1}^{C_i} (p_{m,n,c_i}) \tag{4}$$

IV. ORTHRUSPE

OrthrusPE is a single processing element which operates in two modes, namely the binary and fixed-precision modes. In the binary mode, OrthrusPE executes SIMD binary Hadamard products and the subsequent popcount-accumulation operations in Eq. 3. The fixed-precision mode is activated for the scaling and accumulation operations shown in Eq. 2 and Eq. 4 as well as shifting, batch normalization and non-binary layers. OrthrusPE leverages a single DSP, which is reconfigured at runtime, to achieve efficient execution in both modes of operation.

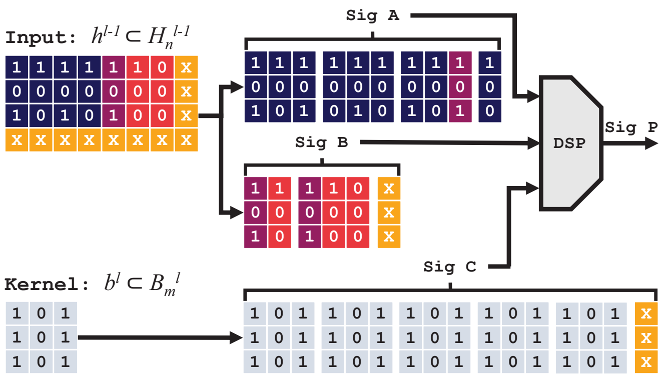
Input: $h^{l-1} \subset H_n^{l-1}$
Sig A
Sig B
Sig C
Sig P
DSP
Kernel: $b^l \subset B_m^l$

Fig. 1. Preconditioning signals A, B and C to compute five 3×3 Hadamard products. Pixels represented with an X are not relevant for this cycle of operation.

For efficient pipelining and in cases where power and resource utilization are less critical, we propose a static variant of OrthrusPE. Here, we instantiate two DSP blocks in each PE. The first DSP is fixed in our proposed binary mode while the second operates as a typical DSP for executing the fixed-point operations. We refer to this proposed implementation as OrthrusPE-DS (Dual-Static).

The following subsections elaborate on the method of execution for both modes in OrthrusPE.

A. SIMD Binary Hadamard Product in Binary Mode

In the example dataflow shown in Fig. 1, OrthrusPE receives two tensor slices, h^{l-1} and b^l. When the kernel b^l is slid over the partial input feature map h^{l-1}, a new tensor slice from H^{l-1}_n can be read from the memory.

The DSP48 (DSP48E1) slice is presented in Fig. 2. We exploit the concatenation of signals A and B to fit part of the tensor slice h^{l-1} and select it through the X multiplexer, forming a 48-bit wide signal. The DSP's multiplier needs to be set to its dynamic mode to allow the use of the concatenated A:B signal, as well as the individual A and B signals during regular multiplication. The multiplier in the DSP is asymmetric, as signals A and B are 30 bits and 18 bits respectively. These signals are preconditioned before entering the DSP, such that their concatenated value represents multiple sliding windows of h^{l-1}. The tensor slice b^l, in the form of Signal C, is available at multiplexers Y and Z (as well as W on DSP48E2). Signal C is preconditioned to hold the b^l

kernel which is to be operated with the h^{l-1} pixels. By setting the ALUMODE signal and activating the correct multiplexers through OPMODE, the DSP is essentially transformed into a SIMD binary Hadamard product module. Note that there exist multiple combinations of ALUMODE and OPMODE states that provide the same result.

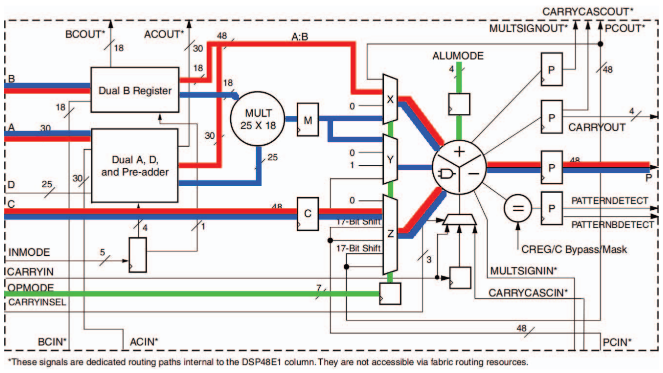

Fig. 2. The DSP48E1 Slice [27]. Appended bold paths illustrate the relevant signals for our operating modes. Red indicates binary mode, blue for fixed-precision mode and green for reconfiguration.

Considering that the most frequently occurring kernel size in modern CNNs is 3×3 pixels [2], each cycle generates five individual Hadamard products at the output of the DSP. In this case, each Hadamard product requires 9 bits out of the 48 bits in signal C and signal A:B, leaving 3 unused bits after the calculation of the 5 results. However, this still presents a high utilization of 94%. This is due to the fact that we can fully fit $\lfloor 48/N \rfloor$ Hadamard products in a single DSP cycle's output, where N is the number of bits per Hadamard product. The solution to always retain a utilization of 100% is to allow partial operations to take place in each cycle, while small additional logic rearranges the successive results before being processed by the popcount logic. In this manner, any arbitrary window size can also be implemented with 100% utilization. Intuitively, for fully connected layers, the utilization is always 100%. Fig. 3 shows the DSP utilization for two possible configurations of OrthrusPE.

In our experiments, preconditioning the signals, concatenating them and performing the wide XNOR operation did not infer a DSP slice, but rather generated a regular LUT solution. Therefore, the DSP slice was instantiated manually and the signals were explicitly passed to the module.

B. Arithmetic Operations in Fixed-Precision Mode

We have already established that contrary to what the name might suggest, accurate binary neural networks involve many regular fixed-point arithmetic operations. As depicted in Eq. 3, after a binary Hadamard product is calculated, its popcount reduces it to a single fixed-point value. All subsequent calculations required for an output pixel which involve this popcount are carried out in fixed-point arithmetic. Therefore, full reliance on binary operations is infeasible, even in naive, inaccurate binary networks. The further scaling operations required in accurate binary nets also introduce fixed-point

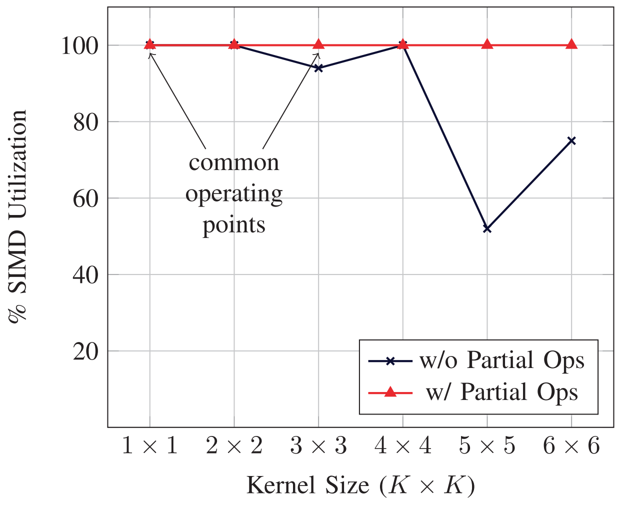

Fig. 3. SIMD register utilization with and without partial operations.

multiplications, which are most efficiently executed on DSPs. This is the motivation for reconfiguring the DSP back to its regular operation mode, by resetting the ALUMODE and OPMODE signals at runtime. With this solution, we exploit the same hardware resource for two distinct modes of operation.

As mentioned in section II, the work done by Double MAC [23] can be appended to OrthrusPE, giving it a further mode to operate in. This is particularly useful for accumulating popcounts as they have a smaller bit width compared to the fixed-point values used for the non-binarized inputs of the network.

C. Mode Switching and Partial Sum Accumulation

While in binary Hadamard SIMD mode, OrthrusPE can generate five 3×3 Hadamard products at a time. These products are passed to popcount logic, generating five fixed-point values. Referring back to section III, each of the M binary weight bases needs to be convolved with each of the N binary activation bases. A single activation channel and a single filter channel produce $M \times N$ partial sum maps. Those $M \times N$ maps need to be scaled by α and β, then collapsed into a single partial sum map. The $M \times N$ maps represent parasitic partial sums which will require a considerable amount of memory if not accumulated for an extended time during execution. To minimize this, OrthrusPE can perform $P \times M \times N$ Hadamard products, where P is a set of p_{m,n,c_i} pixels, then switch to its MAC mode for scaling and accumulation, before moving on to another spatial region of the map. Decreasing P reduces the required memory for parasitic partial sums as shown in Eq. 5. In the last term, the kernel dimensions dictate the popcount's bit width along with an added sign-bit.

$$Mem_{psum} = N \times M \times P \times (\lceil log_2(K \cdot K) \rceil + 1) \qquad (5)$$

The trade-off is that accumulating the $P \times M \times N$ popcounts requires switching the mode of OrthrusPE more often. The number of mode switches per input channel map is expressed in Eq. 6. X_o and Y_o are the dimensions of a single output channel. Since it is possible to switch the ALUMODE after 1 cycle of operation for non-pipelined DSPs, this trade-off does not represent a large overhead and can be exploited to reduce partial result memory requirements in an accelerator.

$$SwitchCount = 2 \times \left\lceil \frac{X_o \cdot Y_o}{P} \right\rceil - 1 \tag{6}$$

P can be chosen with some analysis using Eq. 5 and Eq. 6. Fig. 4 shows the effect of P on partial result memory and switch count for a single input channel of binary ResNet-18 [2] in each of its convolutional layers, with $M = N = 3$. The layers are grouped based on their output spatial dimensions.

Layer 1 is not considered as it is not binarized. The actual scratchpad size depends on other factors such as dataflow, loop unrolling and loop interleaving. The analysis shown is only with respect to the minimum requirement necessary for a basic dataflow which maintains the partial results within a PE until a single input channel is completely processed against a single filter kernel.

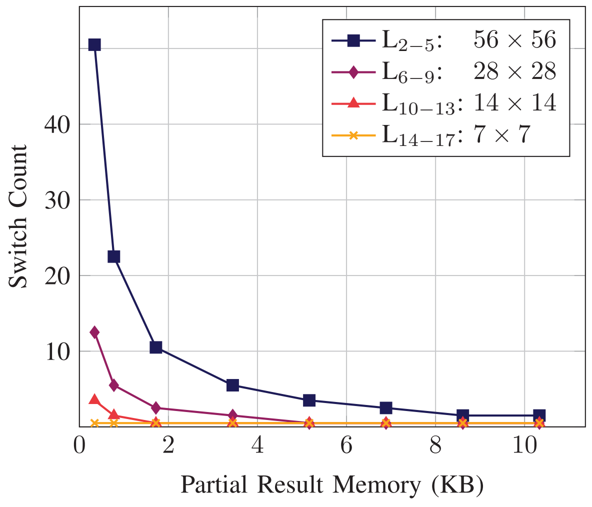

Fig. 4. Switch count and partial result memory analysis for a single input channel from different convolutional layers of binary ResNet-18, with $M = 3$, $N = 3$. Each point represents a different configuration of P.

V. Evaluation

A. Experimental Setup

We compare the proposed implementations in section IV to two implementations with equivalent functionality. Typically, BNN processing elements employ two or more distinct types of resources for the operations described in Tab. I. On FPGAs, the straightforward approach is to map all binary operations to LUTs and execute the supported fixed-point operations on DSPs. This translates to a single PE execution spanning two different types of hardware resources. We refer to this implementation as the "Hybrid" implementation. For completeness, we compare a fourth implementation that restricts execution of the operations to the FPGA's LUT resources.

All four implementations were synthesized and implemented using the Xilinx Vivado 2018.1 synthesis tool targeting the Zynq UltraScale+ MPSoC ZCU102. Correct functionality of OrthrusPE was confirmed by the Xilinx Vivado Simulator. Power estimates are obtained using the Xilinx Power Estimator (XPE) and the Vivado Power Analysis tool, built into the Vivado Design Suite.

In order to fairly compare the four implementations, we fix the throughput to 1 MAC per cycle or 48 XNORs per cycle, i.e. a single OrthrusPE's throughput. Higher performance of all implementations is possible by replicating the structures. The processing elements were synthesized across multiple target frequencies to show compatibility with any potential accelerator which might utilize them. A BNN accelerator would typically require *hundreds* of PEs, therefore, the presented results scale gracefully based on the underlying accelerator.

B. Resource Utilization Analysis

Implementation of the individual PEs yielded the utilization results presented in Table II.

TABLE II
Resource Utilization results of the tested implementations.

Implementation	F=770MHz			F=160MHz		
	LUTs	FF	DSP	LUTs	FF	DSP
All-LUT	559	160	0	516	160	0
Hybrid (Common)	230	253	1	166	253	1
OrthrusPE	**165**	210	1	**111**	210	1
OrthrusPE-DS	**120**	229	2	**87**	229	2

The results show that OrthrusPE can operate at the maximum frequency of the DSP48 block, i.e. the added functionality comes without any cost of latency. Practically BNN accelerators operate at lower frequencies, therefore OrthrusPE can be implemented on any BNN FPGA accelerator.

Fig. 5 shows that at high frequency designs the utilization differences within each implementation are minimal. However, from 500MHz down to 400MHz, three of the implementations enjoy some relaxation in the parallelism required to meet the timing constraints. Another such relaxation occurs when the target frequency is lowered from 400 MHz down to 333 MHz. Overall, our OrthrusPE and OrthrusPE-DS implementations, both executing SIMD binary Hadamard product on DSPs, result in the lowest LUT utilization cost.

A further plot point is added showing the resource utilization quoted in FINN [19] for popcount-accumulation of 128-bits at a target frequency of 200MHz. The closest matching OrthrusPE implementation, in terms of bit width, provides 16 more bit accumulations and 3 parallel MAC operations (through runtime reconfigurability), while requiring 32% fewer LUTs.

C. Dynamic Power Analysis

Fig. 6 shows the power estimates for the implementations at different design taget frequencies. The results demonstrate that using a single OrthrusPE for MAC operations and binary Hadamard products presents the most efficient solution among the evaluated implementations. The OrthrusPE-DS solution also offers the second best power efficiency among the 4

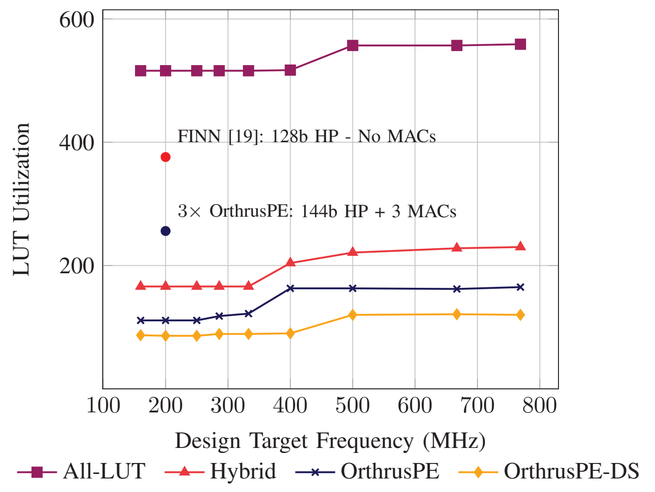

Fig. 5. Synthesis results for LUT utilization across different design target frequencies. Each plot point represents a different synthesis run.

configurations. In practice, OrthrusPE-DS can execute both types of operations concurrently which makes it well-suited for a pipelined accelerator.

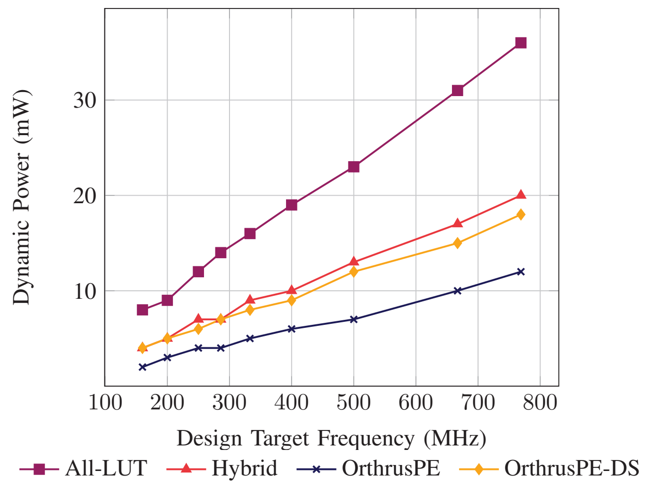

Fig. 6. Dynamic power estimation at different design target frequencies. Each plot point represents a different synthesis run.

The results show that DSPs present a good choice for accelerating binary operations in OrthrusPE and OrthrusPE-DS. Default implementations relying purely on LUTs or hybrids of LUTs and DSPs were less efficient in all of our experiments. Exploiting DSPs as in OrthrusPE improves the utilization of hard blocks already employed by accurate BNN accelerators. This does not prevent the design from employing further LUTs for further binary operations, yet it allows hard blocks to contribute to more types of computations.

VI. Conclusion

In this paper we focus on the efficient execution of BNNs at the compute level. We propose a runtime reconfigurable PE which can satisfy all the functions required by accurate BNNs, while capitalizing on resource reuse. Accurate BNNs cannot be achieved without fixed-point operations and reliance on DSP blocks. Instead of separating binary and fixed-point computations to two types of hardware resources, OrthrusPE improves the efficiency of the computation by executing both on FPGA hard blocks. We evaluated OrthrusPE and OrthrusPE-DS across multiple target accelerator frequencies. Both solutions achieved improved resource utilization *and* power efficiency compared to typical BNN accelerator processing elements. Accurate BNNs solve many of the computation and memory challenges for deep neural network workloads on edge devices. Efficiently executing their mixed-precision computations can further exploit the advantages they offer at the hardware level.

References

[1] O. Russakovsky *et al.*, "Imagenet large scale visual recognition challenge," *Int. J. Comput. Vis.*, Dec. 2015.
[2] K. He *et al.*, "Deep residual learning for image recognition," in *CVPR*, Jun. 2016.
[3] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *CVPR*, Jun. 2015.
[4] M. R. Vemparala *et al.*, "An efficient fpga accelerator design for optimized cnns using opencl," in *ARCS*, 2019.
[5] A. Frickenstein, C. Unger, and W. Stechele, "Resource-aware optimization of dnns for embedded applications," in *CRV*, May 2019.
[6] V. Sze *et al.*, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, Dec. 2017.
[7] A. Frickenstein *et al.*, "DSC: Dense-sparse convolution for vectorized inference of convolutional neural networks," in *CVPR-W*, June 2019.
[8] I. Hubara *et al.*, "Binarized neural networks," in *NIPS*, 2016.
[9] M. Courbariaux, Y. Bengio, and J. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *NIPS*, 2015.
[10] S. Darabi *et al.*, "Bnn+: Improved binary network training," *CoRR*, 2018.
[11] X. Lin *et al.*, "Towards accurate binary convolutional neural network," in *NIPS*, 2017.
[12] M. Rastegari *et al.*, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *ECCV*, 2016.
[13] H. Sharma *et al.*, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural networks," in *ISCA*, 2018.
[14] J. Lee *et al.*, "Unpu: An energy-efficient deep neural network accelerator with fully variable weight bit precision," *IEEE J. Solid-State Circuits*, Jan. 2019.
[15] P. Judd *et al.*, "Stripes: Bit-serial deep neural network computing," in *MICRO*, Oct. 2016.
[16] S. Sharify *et al.*, "Loom: Exploiting weight and activation precisions to accelerate convolutional neural networks," in *DAC*, 2018.
[17] R. Andri *et al.*, "Yodann: An architecture for ultralow power binary-weight cnn acceleration," *IEEE TCAD*, Jan. 2018.
[18] K. Ando *et al.*, "Brein memory: A single-chip binary/ternary reconfigurable in-memory deep neural network accelerator achieving 1.4 tops at 0.6 w," *IEEE J. of Solid-State Circuits*, Apr. 2018.
[19] Y. Umuroglu *et al.*, "Finn: A framework for fast, scalable binarized neural network inference," in *FPGA*, 2017.
[20] R. Zhao *et al.*, "Accelerating binarized convolutional neural networks with software-programmable fpgas," in *FPGA*, 2017.
[21] L. Yang, Z. He, and D. Fan, "A fully onchip binarized convolutional neural network fpga impelmentation with accurate inference," in *ISLPED*, 2018.
[22] S. Liang *et al.*, "Fp-bnn: Binarized neural network on fpga," *Neurocomputing*, vol. 275, 2018.
[23] D. Nguyen, D. Kim, and J. Lee, "Double mac: Doubling the performance of convolutional neural networks on modern fpgas," in *DATE*, Mar. 2017.
[24] E. Nurvitadhi *et al.*, "Accelerating binarized neural networks: Comparison of fpga, cpu, gpu, and asic," in *FPT*, Dec. 2016.
[25] Xilinx, Inc, *Versal: The First Adaptive Compute Acceleration Platform (ACAP)*, 10 2018. v1.0.
[26] W. Tang, G. Hua, and L. Wang, "How to train a compact binary neural network with high accuracy?," in *AAAI*, 2017.
[27] Xilinx, Inc, *7 Series DSP48E1 Slice*, 3 2018. v1.10.