Embracing Diversity: Enhanced DSP Blocks for Low-Precision Deep Learning on FPGAs

Andrew Boutros*†, Sadegh Yazdanshenas*, and Vaughn Betz*†
*Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada
†Vector Institute, Toronto, ON, Canada
E-mails: {andrew.boutros, sadegh.yazdanshenas}@mail.utoronto.ca, vaughn@eecg.utoronto.ca

***Abstract*—Use of reduced precisions in Deep Learning (DL) inference tasks has recently been shown to significantly improve accelerator performance and greatly reduce both model memory footprint and the required external memory bandwidth. With appropriate network retuning, reduced precision networks can achieve accuracy close or equal to that of full-precision floating-point models. Given the wide spectrum of precisions used in DL inference, FPGAs' ability to create custom bit-width datapaths gives them an advantage over other acceleration platforms in this domain. However, the embedded DSP blocks in the latest Intel and Xilinx FPGAs do not natively support precisions below 18-bit and thus can not efficiently pack low-precision multiplications, leaving the DSP blocks under-utilized. In this work, we present an enhanced DSP block that can efficiently pack 2× as many 9-bit and 4× as many 4-bit multiplications compared to the baseline Arria-10-like DSP block at the cost of 12% block area overhead which leads to only 0.6% total FPGA core area increase. We quantify the performance gains of using this enhanced DSP block in two state-of-the-art convolutional neural network accelerators on three different models: AlexNet, VGG-16, and ResNet-50. On average, the new DSP block enhanced the computational performance of the 8-bit and 4-bit accelerators by 1.32× and 1.6× and at the same time reduced the utilized chip area by 15% and 30% respectively.**

I. Introduction

Deep neural networks (DNNs) have been widely replacing conventional machine learning algorithms as the *de facto* standard in numerous applications. However, their unprecedented accuracy and ability to learn complex features from unstructured data comes at the cost of increased computational complexity and a large memory footprint. The compute and memory requirements further escalate as models get deeper and more complex in order to achieve higher accuracy. For example, model size increased by a factor of 2.3× in VGG-16 compared to AlexNet for an 11% increase in top-1 image classification accuracy [1]. As a result, many researchers are investigating the use of reduced precision weights and/or activations, especially in inference tasks that do not necessarily require full-precision floating-point arithmetic [2].

A linear reduction of 4× in model memory footprint and required external memory bandwidth can be achieved when using 8-bit fixed-point instead of 32-bit floating-point activations and weights for example. Although these gains come at the cost of accuracy degradation, the resilience of deep learning (DL) models to noise made it possible to use 16-bit fixed-point arithmetic with less than 0.5% accuracy degradation on the ImageNet dataset [3]. Several techniques, such as incremental quantization [4] and increasing the depth of convolutional layers [5], enabled the use of further lower precisions ranging from 8-bit and 4-bit down to ternary and binary with negligible or no accuracy degradation.

The advances in mitigating the accuracy degradation when using reduced-precision DNNs have also led hardware architects to add support for low-precision arithmetic units into their DL inference accelerators. In addition to reducing memory footprint and the required external memory bandwidth, this also allows fitting more multiply-accumulates (MACs) on-chip and achieving higher energy efficiency. The gains can be substantial: a 16-bit fixed-point multiplication consumes about 6.2× less energy than a 32-bit floating-point multiplication in 45nm process technology [3]. Examples of architectures with low-precision support include Nvidia's Tesla P40 DL inference accelerator, which added support for 8-bit operations resulting in a 4× higher inference throughput compared to their equivalent 32-bit floating-point architectures [6], and Google's TPU which includes an 8-bit 256×256 matrix multiply-accumulate unit to accelerate inference workloads in datacenters [7].

On the other hand, FPGAs are an attractive low-precision DNN acceleration option due to their: flexibility in building custom bit-width datapaths, higher energy efficiency compared to GPUs, and lower non-recurring engineering cost compared to ASICs. However, the digital signal processing (DSP) blocks in the latest Intel and Xilinx FPGAs fall short on flexibility as they do not natively support multiplications smaller than 18 bits. Therefore, when implementing low-precision DL accelerators, designers resort to either implementing most of the small bit-width multipliers in the soft fabric or packing two multiplications that have a common multiplicand into a single multiplier as shown in [8]. Both approaches are suboptimal, leaving DSP blocks underutilized, with the former leading to higher power consumption and possibly lower operating frequency, and the latter imposing the restrictions of having a shared multiplicand and a limited number of MACs before requiring additional logic to adjust the accumulation result.

In this paper, we make the following contributions:

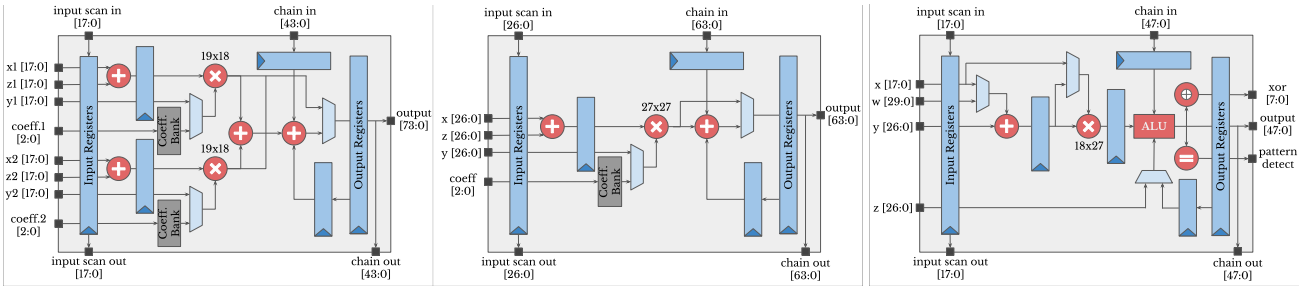
- Enhancing the DSP block on current commercial FPGAs to support 9-bit and 4-bit multiplications/MACs at the cost of minimal area overhead and no frequency drop.
- Quantifying the performance gains of the enhanced DSP block in two state-of-the-art convolutional neural network (CNN) accelerators on a variety of models for both 8-bit and 4-bit precisions.

II. Background and Related Work

A. The Chaos of Deep Learning Precisions

The idea of using low precisions in neural networks goes back to the early 1990s when Holt and Baker compared the use of floating-point and limited-precision integer back propagation to build efficient neural network hardware [9]. After the recent resurgence of interest in DNNs, research was initially focused

978-1-5386-8517-4/18/$31.00 ©2018 IEEE
DOI 10.1109/FPL.2018.00014



(a) Intel's DSP block (dual 18×19 mode) (b) Intel's DSP block (27×27 mode) (c) Xilinx's DSP block

Fig. 1: DSP blocks on current FPGAs.

on maximizing model accuracy and achieving near human-level accuracy in some applications. The resulting large models can be challenging to deploy both in embedded mobile applications and large-scale datacenters that have tight constraints on latency and power budget. Therefore, many recent works have focused on co-designing DL models and the underlying computing hardware to improve efficiency by reducing the arithmetic precision without affecting the hard-won accuracy.

Some of the early works in this area focused on training simple CNNs on MNIST and CIFAR-10 datasets using low numeric precisions like 16-bit weights and activations [10], 10-bit weights and 12-bit activations [11] down to 2-bit weights and 3-bit activations [12]. In [2], the authors introduced Ristretto, a framework for quantizing floating-point CNNs using dynamic fixed-point precision. They were able to use 8-bit weights and activations in several ImageNet models with less than 1% accuracy degradation. Mellempudi et al. [13] used a cluster-based quantization approach on ResNet-101 model that enabled the use of 8-bit activations and 4-bit weights with less than 2% accuracy degradation. Researchers have also investigated several techniques to re-gain the accuracy lost as a result of using low-precision arithmetic. For example, authors of [5] introduced Wide Reduced-Precision Networks to compensate the accuracy loss by increasing the number of filters in each layer when using 8-bit and 4-bit down to ternary and binary weights and activations. The huge variety of precisions used in DL inference is not just limited to reduced bit-widths, but also extends to using non-standard numeric representations such as Microsoft's 9-bit floating-point representation in the FPGA-based project BrainWave [14]. In this project, custom floating point units are implemented in the soft fabric including 4-bit or 5-bit multipliers for mantissa multiplication.

B. DSP Blocks on Current FPGAs

One of the unique advantages of FPGAs has always been the ability to offer a flexible and efficient hardware acceleration solution for domains where various standards are adopted. This is the current case for DL inference in which the optimal precision is still undecided and model-dependent as discussed in the previous subsection. Unlike CPUs and GPUs, FPGAs can benefit from reducing the data-path width either by reducing the power consumption or flexibly fitting more computations on-chip. DSP blocks in Intel's latest FPGAs [15] can pack either two 18×19 multiplications or one 27×27 multiplication per DSP block as shown in Fig. 1(a) and 1(b) respectively. They support different modes such as independent multiplications, pre-addition, and multiply-accumulate as well as having dedicated input and output chains to implement efficient vector structures for applications such as FIR filters, dot-products and matrix multiplication. Intel has also introduced support for single-precision floating-point arithmetic in the DSPs of Arria 10 FPGAs with only 10% block area overhead [16]. On the other hand, DSP blocks in Xilinx FPGAs, shown in Fig. 1(c), support one 18×27 multiplication and contain a 48-bit Arithmetic Logic Unit, a wide XOR operator, a pattern detector, and dedicated input and output chains [17]. One unique feature in Xilinx DSP blocks is that their operating mode is dynamically controlled by external signals at runtime in contrast to Intel's DSP blocks that are statically configured at compilation time using SRAM cells. Unlike Intel and Xilinx, DSP blocks in Microsemi FPGAs support two 9-bit multiplications in the dot-product mode and a single 18-bit multiplication in the normal mode [18].

C. Related Work on Multi-precision DSP blocks

In [19], the authors proposed a new DSP block architecture that supports different multiplication bit-widths and exposes the partial product reduction logic to the designer to implement efficient multi-input addition. Their proposed DSP block added support for eight 9-bit, six 12-bit, two 24-bit and one 36-bit for an 11% area overhead and an additional 2% for supporting multi-input addition compared to a base DSP block that only implements four 18-bit multiplications. They showed that their multi-input addition is on average 50% faster than that implemented using soft logic. However, they did not present any study of benchmark circuits that benefited from supporting 12-bit and 24-bit multiplications that were not supported in the equivalent Stratix-II DSP block. Also, the authors do not mention any details about implementing the MAC mode in their DSP block which is necessary for the vector operations and dot-products that are heavily used in DL applications. The authors of [20] proposed adding runtime fracturability to the 16-bit DSP blocks in a multi-context reconfigurable architecture. Their new DSP block added support for two 8-bit multiplications at the cost of 13% area overhead and an extra pipeline stage to reduce the critical path delay. In both studies, the reported area for the enhanced DSP block is only based on synthesis results which can be misleading as we will show in the results section. In our study, we also present a detailed evaluation of the performance gains of using our enhanced DSP block in several reduced-precision CNN benchmarks.

III. DSP Block Architecture

In this section, we first discuss some arithmetic preliminaries necessary for understanding the rest of this section and then describe the baseline Arria-10-like DSP block architecture followed by implementation details of our enhanced DSP block to support 9-bit and 4-bit multiplications.

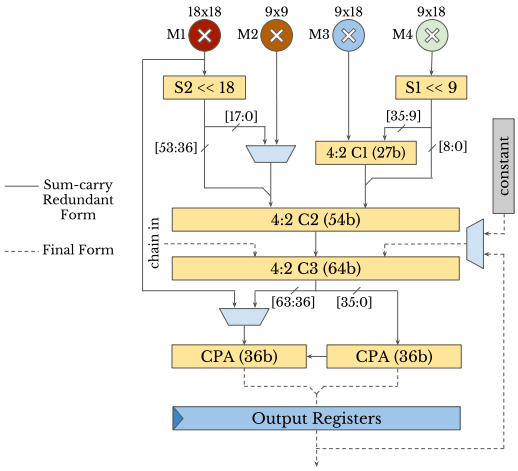
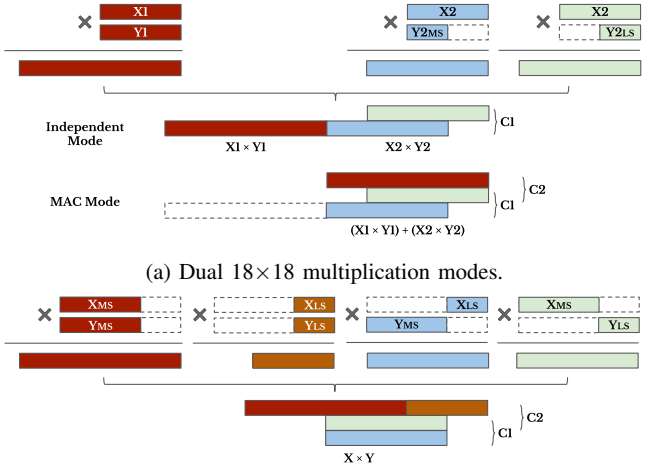


Fig. 2: Arria-10-like baseline DSP block architecture.



(a) Dual 18×18 multiplication modes.

(b) 27×27 multiplication mode.

Fig. 3: Mapping of different modes to the baseline DSP block architecture.

A. Arithmetic Preliminaries

The partial product generation of the multipliers in our proposed DSP block is based on the *Baugh-Wooley Algorithm* for two's complement signed multiplication [21] followed by a *Dadda Reduction Tree* [22]. We developed a script that generates the Verilog hardware description for the partial product generation and reduction given any multiplier size. The output of each multiplier is represented in redundant form as a sum and a carry that are summed up only immediately before exiting the DSP block to avoid the delay of multiple *carry propagate adders (CPAs)* in modes like sum-of-products or multiply-accumulate. All intermediate additions in the DSP block are performed using *4:2 compressors*. A single-bit 4:2 compressor takes 4 input bits and a carry-in bit to output a sum bit, a carry bit of higher significance and a carry-out bit that propagates to the subsequent bit compressor. The advantage of 4:2 compressors is that the carry-out is only dependent on the inputs and not the carry-in which eliminates any carry propagation delays.

B. Baseline DSP Block

In this work, we implement an Arria-10-like DSP block as a baseline to which we later compare our modified DSP block. This baseline DSP supports one 27-bit multiplication that can be fractured into two 18-bit multiplications. This DSP block is similar to that in Arria 10 and Stratix 10 chips but without floating-point arithmetic support which is out of the scope of this work. Fig. 2 shows the architecture of the DSP block we implement based on [23]. It contains four different multipliers: one 18×18 ($M1$), one 9×9 ($M2$), and two 9×18 ($M3$, $M4$) multipliers. The multipliers are followed by a series of shifters ($S1$, $S2$) and compressors ($C1$, $C2$, $C3$) to implement different DSP block modes. Notice that the solid lines in Fig. 2 represent numbers in sum-carry redundant form that are summed using the final CPA.

In the dual 18-bit multiplication mode, $M1$ is used to multiply the first two operands, X_1 and Y_1, while the second multiplication is performed using $M3$ and $M4$. The operand Y_2 is split into two 9-bit halves, as illustrated in Fig. 3(a), where each of the most significant half ($Y_{2_{MS}}$) and the least significant half ($Y_{2_{LS}}$) is multiplied by X_2 on $M3$ and $M4$ respectively. In this mode, $M2$ is idle, the shifter $S1$ is bypassed and the compressor $C1$ is used to add the two partial results of the second multiplication. In the case of two independent 18×18 multiplications, the outputs of $M1$ bypass all the intermediate blocks and are added separately in the upper half of the final CPA, while the lower half of the CPA is used to add the outputs of the second multiplication after passing through the compressors $C2$ and $C3$. In the case of MAC shown in Fig. 3(a), the shifter $S2$ is bypassed and the compressor $C2$ is used to add the two multiplication results before adding it to the input chain and/or the previous cycle result using $C3$. The two CPAs are cascaded by propagating the carry-out of the right CPA to the carry-in of the left one.

In the 27-bit multiplication mode, each of the two multiplicands, X and Y, are split into two parts: the most significant 18 bits and the least significant 9 bits. $M1$ and $M2$ are used to perform the two symmetric multiplications $X_{MS} \times Y_{MS}$ and $X_{LS} \times Y_{LS}$. The two asymmetric multipliers $M3$ and $M4$ are used to multiply $Y_{MS} \times X_{LS}$ and $X_{MS} \times Y_{LS}$ respectively. The output of $M1$ is shifted to the left by 18 bits using $S2$ and concatenated with the output of $M2$ while the output of $M4$ is shifted to the left by 9 bits using $S1$ and added to the output of $M3$ using the compressor $C1$. The two partial outputs are then added using $C2$ as shown in Fig. 3(b) to get the final result of the 27×27 multiplication which can be accumulated using $C3$ in a similar manner to the 18-bit MAC mode.

When changing the baseline DSP to support lower precision multiplications, we set the following specifications:

- All the modes in the baseline DSP block must be supported in the modified DSP block to ensure backward compatibility and avoid negatively affecting other benchmark applications.
- The number of input and output ports must not increase to minimize the area overhead and avoid stressing the routing to and from the DSP block.
- The modified DSP block must have an operating frequency of 600 MHz, matching that of the commercial DSP block in the same 28nm technology node which we use in our study. [24].

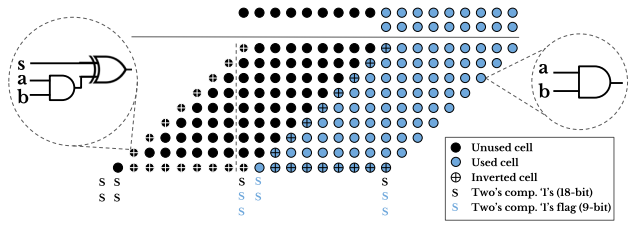


Fig. 4: Modifications to the $M4$ multiplier array to support 9-bit multiplication.

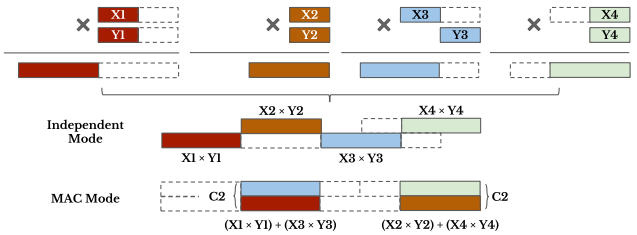


Fig. 5: Mapping of 9-bit independent multiplies and MAC to the modified DSP block architecture.

C. Adding Support for 9×9 Multiply and MAC modes

Since the baseline DSP block has 72 output ports, it can pack at most four 9-bit multiplications. We use $M1$ to perform the first multiplication by feeding the multiplicands to the most-significant 9 bits of its inputs and obtaining the result from the most-significant 18 bits of its output. We also use the 9×9 multiplier $M2$ to perform the second multiplication and modify the two other multipliers ($M3$, $M4$) to achieve correct functionality for signed multiplications. For $M4$, as shown in Fig. 4, we enable only the lower half of the partial product generation array in the 9-bit multiplication mode. This requires adding inverting capability to the cells on the left and bottom borders of the used array as well as the two's complement '1's (denoted by S) in the locations shown in Fig. 4 to correctly implement the Baugh-Wooley multiplication algorithm. The carries in the partial product reduction tree are also killed so as not to propagate beyond the dashed line and ensure that the highest-significant 9 bits of the result are zeros in this mode. We follow the same approach in $M3$ but for the higher half of the partial product generation array to ensure that the results of $M3$ and $M4$ do not overlap when added in $C1$. For $M3$, the inverting cells are already present and we only need to add the two's complement '1's in their proper location. Fig. 5 illustrates how the four 9-bit multiplications are performed in the modified DSP block. The least-significant 18 bits of the output of $M1$ are replaced by the result of $M2$. For the other two results, the shifter $S1$ is bypassed and the compressor $C1$ is modified to kill the carry propagation after its first 9 bits to separate the two results of $M3$ and $M4$.

For the 9-bit MAC mode, adding all four results would require an additional compressor stage between $C2$ and $C3$. When we implemented this, the design failed to meet the 600 MHz timing constraint due to the longer critical path through three stages of compressors. Alternatively, we decided to split the MAC operation such that each two of the four multiplication results are added together producing two results

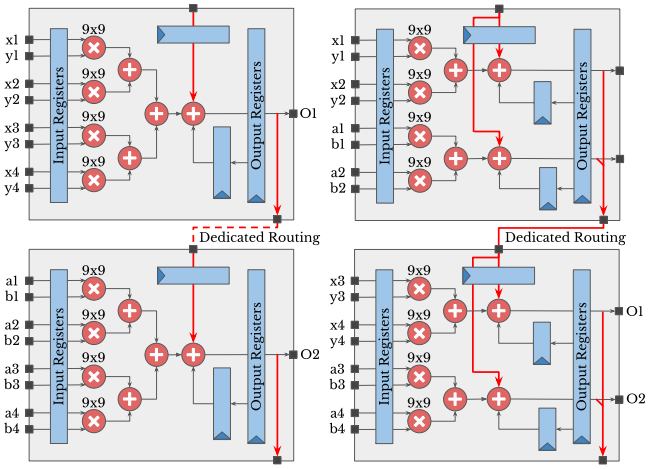


Fig. 6: Splitting the 9-bit MAC mode (right) to avoid second addition stage (left).

instead of one. In this case, two dot-product units of size 4 are mapped to two DSP blocks cascaded using the dedicated routing chains instead of one dot-product unit per DSP block as shown in Fig. 6. In this mode, the compressor $C2$ adds the outputs of $M2$ and $M4$ in the least-significant 36 bits as well as the outputs of $M1$ and $M3$ in the higher significance bits as shown in Fig. 5. This approach eliminates the second layer of addition, does not require any increase in the number of dedicated routing chain wires, provides the same density of MACs and gives 10 extra bits for accumulation per result similar to that in the 27-bit MAC mode.

D. Adding Support for 4×4 Multiply and MAC modes

Adding support for eight 4-bit multiplications is more challenging since there are many different design choices to explore. One approach is to further fracture the four available multiplier arrays such that each one accommodates two 4-bit multiplications by dividing it into four quadrants and adding masking logic to the unused ones and signed multiplication logic to the used ones. Another approach is to add standalone 4×4 multipliers that have a small area footprint to the DSP block. We implemented the three variations listed below to find the design point that has the least area overhead while meeting the 600 MHz timing constraint: (1) Fracturing the four multipliers to perform two 4-bit multipliers each. (2) Fracturing the two multipliers $M2$ and $M3$ that are not on the critical path of the DSP block and avoiding fracturing the other two multipliers. From Fig. 2, we can see that both $M1$ and $M4$ pass through more layers of shifters and compressors and any addition of fracturing logic will lengthen the critical path. We add four standalone 4×4 multipliers in this case. (3) Fracturing the two multipliers $M2$ and $M3$, adding signed multiplication support to use $M1$ and $M4$ to perform one 4-bit multiplication each and adding two standalone 4×4 multipliers. Interestingly, our experiments showed that the second approach had the least area overhead as will be detailed in Section IV, even though it left both $M1$ and $M4$ idle and added four small multipliers. Fig. 7 shows how the eight 4×4 multiplications are performed in the DSP block. Four multiplications are performed using the four standalone multipliers ($M5 - M8$) while the other four are performed on

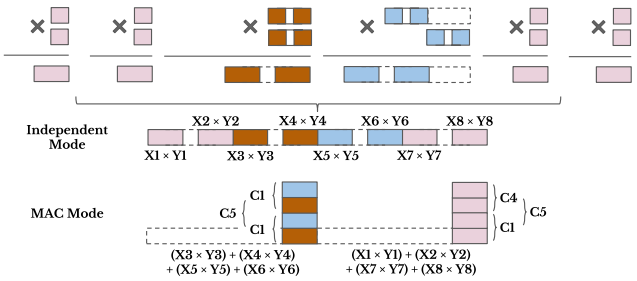


Fig. 7: Mapping of 4-bit independent multiplies and MAC to the modified DSP block architecture.

the fractured $M2$ and $M3$. Since $M2$ and the used half of $M3$ are of size 9×9, the two 4-bit inputs are separated by 1 bit and thus their two results are 2 bits apart.

For the 4×4 MAC mode, we use a similar approach to that used in the 9×9 mode. The results of every four multiplications are added together to produce two outputs that are exposed to the soft-fabric or propagate through chains of DSP blocks. We modify the compressor $C1$ in Fig. 2 to be fracturable into three 9-bit units that are used to sum three pairs of results and we add another 9-bit compressor ($C4$) to sum the fourth pair. We also add another compressor ($C5$) to perform the second level of addition and then multiplex between the outputs of $C2$ and $C5$; this proved to be cheaper and more routing-friendly than adding a multiplexing layer and using $C2$ for this purpose. The final architecture of the enhanced DSP block is shown in Fig. 8.

IV. DSP Block Results

We implement the baseline and enhanced DSP blocks in structural Verilog HDL and perform simulation using ModelSim Starter Edition 10.5b to verify their functionality in all supported modes for corner cases as well as randomly generated input vectors. We use the hybrid COFFE flow described in [25] to obtain the area of the DSP blocks. This flow can model hard blocks in heterogeneous FPGAs such that the core of the block is implemented in standard cells using a given library while the interface to the programmable interconnect (i.e. local routing crossbars, switch boxes, connection boxes and dedicated chain routing buffers) is implemented in full custom. COFFE uses iterative HSPICE simulations to perform transistor sizing of the full custom interface. In the standard-cell flow, it uses 28nm STMicroelectronics libraries (1.0V, 125 °C, worst-case) to first synthesize our designs using Synopsys Design Compiler 2013.03 and then place and route them using Cadence Innovus 16.

Table I shows the area breakdown of our baseline DSP block obtained from COFFE. When scaling down the area of the Stratix-III DSP block in [26] to 28nm, its area is 48673 μm^2 which is $\sim$ 4.9× that of our baseline DSP block. Given that

TABLE I: Breakdown of baseline DSP block area.

Component	Area (μm^2)
Standard-cell Core	7367
Switch Block	828
Connection Block	1042
Local Crossbar	588
Dedicated Chain Buffers	50
Total	**9875**

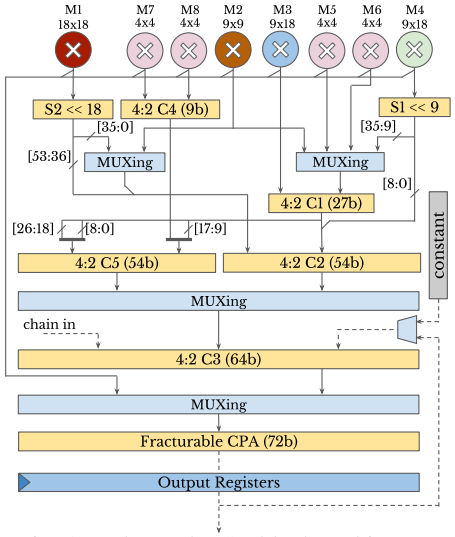


Fig. 8: Enhanced DSP block architecture.

the Stratix-III DSP block contains eight 18-bit multipliers and spans 4 LAB rows compared to only two 18-bit multipliers and 1 LAB row for our baseline Arria-10-like DSP block, we expect an area ratio around 4×. We do not implement all the skip muxing logic required to bypass defective rows [23], so we expect an area ratio slightly higher than 4× and hence 4.9× is logical and provides a sanity check on our area results.

The area results for the enhanced DSP block are presented in Table II. We use the area of placed and routed designs to calculate the area overhead compared to the baseline DSP block. Supporting four 9-bit multiplications caused an overhead of 4% for the modifications to the $M3$ and $M4$ multiplier arrays and the additional control logic. As a result of using the split MAC approach explained in Section III-C, supporting the 9-bit MAC mode required almost no additional circuitry and thus had an additional cost of only 1%. When evaluating the 3 options discussed in Section III-D to add support for 4×4 multiplications, option (1) did not pass timing at place and route despite achieving the desired 600 MHz frequency in synthesis. Although option (3) had slightly less post-synthesis area than (2), option (2) achieved significantly less area overhead after place and route than option (3) with 9% and 18% increase for (2) and (3) respectively. While option (2) adds more logic to the block, it turned out to be more wiring-friendly and easier to route compared to option (3). This highlights that making detailed design choices based on synthesis results, while a common practice, can be misleading.

Proceeding with option (2), area increased by an additional 11% when using the existing compressor $C2$ and adding multiplexing logic to its inputs to perform the second level of addition in the 4×4 MAC mode. However, when adding another compressor $C5$ and multiplexing logic to choose between the outputs of $C2$ and $C5$, the 11% additional overhead was reduced to only 3%. Our final DSP block supporting one 27-bit, two 18-bit, four 9-bit and eight 4-bit multiplications/MACs has an operating frequency of 600 MHz and an overall area overhead of 12% compared to the baseline Arria-10-like DSP block. In DSP-rich FPGAs, DSP blocks

TABLE II: Area of enhanced DSP blocks and overhead of supporting different modes compared to the baseline.

DSP Block	Post-Synth. Area (μm^2)	Post-P&R Area (μm^2)	Area Ratio
Baseline DSP Block	8404	9875	1.00
Add 9×9 Mult.	**8368**	**10320**	**1.04**
Add 9×9 MAC	**8810**	**10384**	**1.05**
Add 4×4 Mult.(1)-max reuse	9571	–	–
Add 4×4 Mult.(2)-min reuse	**9104**	**10752**	**1.09**
Add 4×4 Mult.(3)-mid reuse	8909	11651	1.18
Add 4×4 MAC using $C2$	9543	11887	1.20
Add 4×4 MAC using $C5$	**9389**	**11108**	**1.12**

consume approximately 5% of the FPGA [16] and therefore the 12% block area overhead is equivalent to only a 0.6% increase in the FPGA core area.

V. Benchmarks Results

A. Benchmarks Description

In order to quantify the performance gains of incorporating our enhanced DSP blocks in current FPGA architectures, we implement the computational core of two state-of-the-art FPGA-based CNN inference accelerators: Intel's Deep Learning Accelerator (DLA) [27] and an architecture from Arizona State University (ASU) [28]. We implement all on-chip buffers, convolution engine and pooling engine. In addition, we also implement the control logic to read features and weights from input buffers, perform the required computations and store the results in the output buffers. However, we do not implement external memory interfacing as it is out of scope and irrelevant to our architectural studies. We also extend the two architectures by adding all the blocks required to accelerate three different CNN models: AlexNet, VGG-16 and ResNet-50 to obtain 6 different benchmark circuits that enable a broad study of the performance gains on different models and computing architectures.

All our implementations are written in SystemVerilog and are parameterizable to easily change the accelerator size (i.e. number of processing elements/multipliers and consequently on-chip buffers) as well as their type (either BSC, LRN or ELT equivalent to accelerating VGG-16, AlexNet or ResNet-50) and their activations/weights precisions. Functionality of the designs is verified using RTL-level simulations in which the input buffers are initialized with feature and weight values and the results written in the output buffers are verified for correctness. We are currently extending our benchmark circuits to include a variety of DL architectures and we are planning to release it as an open-source DL benchmark suite for FPGA architecture explorations.

The DLA architecture has an array of Processing Elements (PEs) arranged in a pipelined daisy-chain scheme as shown in Fig. 9. Each of the PEs contain multiple dot-product units that perform the MAC operations as well as local buffers for storing the weights in the convolution layers and features in the fully connected layers. The two sets of stream buffers interchange roles as input and output buffers in convolutional layers whose features can be completely stored on-chip without the need for any external memory transfers. In fully connected layers, the stream buffers are used to store the weights instead. We removed the Winograd Transform optimization from this computing architecture due to its instability in low precisions

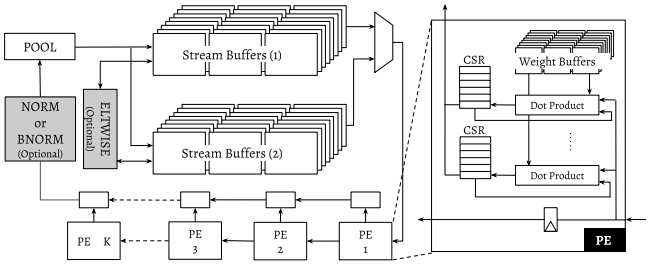


Fig. 9: DLA architecture.

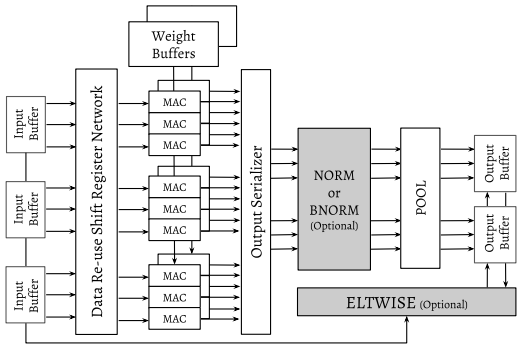


Fig. 10: ASU architecture.

[29] and added support for element-wise (ELTWISE) and batch normalization (BNORM) layers.

The ASU architecture contains distinct input and output buffers as depicted in Fig. 10 while the weights are double buffered in the fully connected layers such that one buffer is filled from external memory while the other is used in computations. This architecture comprises a three dimensional array of PEs preceded by a complex data reuse shift register network that efficiently implements convolutions in a sliding window fashion. We extended this architecture to support the local response normalization (NORM) layer used in AlexNet.

B. Methodology

We synthesize, place and route our designs on the largest and fastest speed grade Arria 10 FPGA (10AX115N2F45I1SG) using Intel Quartus Prime 17. We carry out two sets of experiments: one using 8-bit weights and activations and the other using 4-bit weights and activations. We scale up the size of each accelerator starting from the original sizes (i.e. number of PEs and size of on-chip buffers) published in [27] and [28] for the DLA and ASU architectures until either BRAMs are fully utilized or the design becomes unroutable. If the design consumes all the available DSP blocks, we start implementing multipliers in the soft fabric with the same number of pipeline stages as that used in the DSP blocks. We found that due to the modular nature of the CNN benchmark architectures, performance was insensitive to exactly which multipliers we implemented in the soft fabric. For each size, precision and architecture, we obtain the results when using the original FPGA as well as an FPGA incorporating our enhanced DSP block (i.e. modified FPGA). Since our proposed DSP block has the same routing interface and number of input/output ports as that of the Arria 10 DSP block, we modify our HDL description of the benchmarks by connecting eight 8-bit inputs or sixteen 4-bit

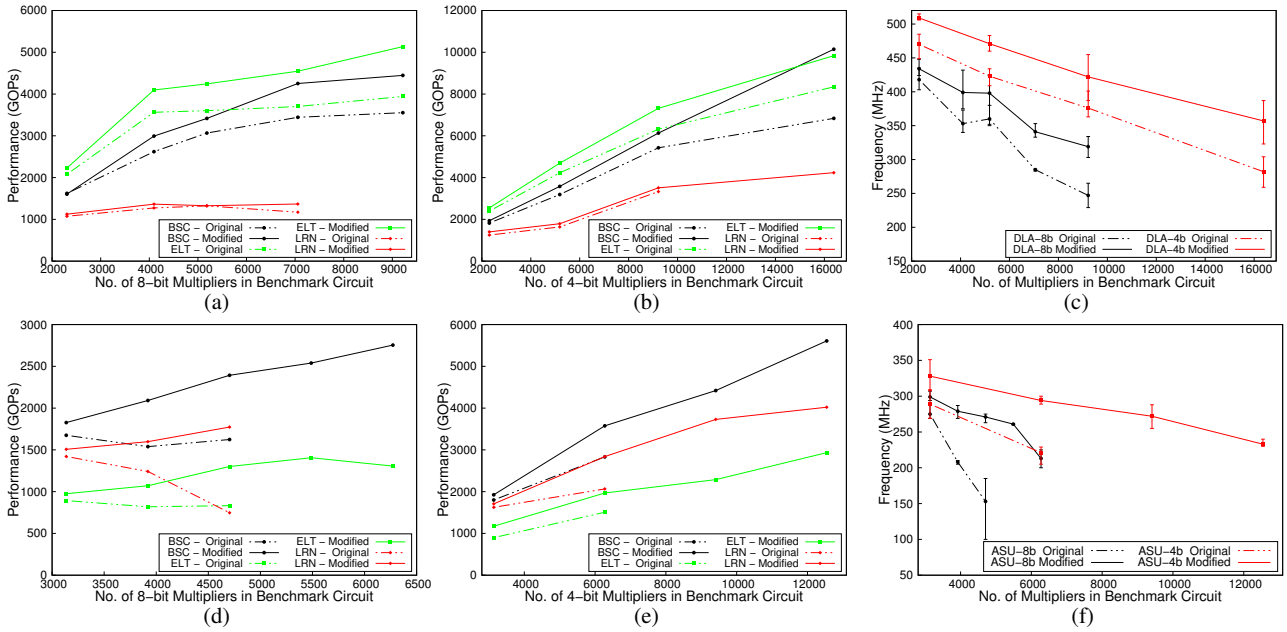


Fig. 11: Performance results using the original and modified FPGAs for different sizes of: (a) 8-bit DLA (b) 4-bit DLA (d) 8-bit ASU (e) 4-bit ASU, and their operating frequencies averaged across BSC, ELT and LRN with error bars indicating maximum and minimum for: (c) DLA (f) ASU.

inputs to each DSP block instantiation in Quartus. This way, we let Quartus route the input and output signals to and from the Arria 10 DSP blocks as if they are our enhanced DSP blocks.

To obtain the performance results, we develop performance models for both benchmark architectures based on our RTL-simulations. The performance models count the number of cycles and the time required for the computation of each layer as well as its external memory transfers according to the tiling and unrolling schemes of each architecture. We assume that the data is laid out in the external memory in a way that allows us to fully use the available memory bandwidth. Our performance models assume two different external memory subsystems: a relatively weak one using single bank of DDR4x64 and a high-end one using the recently introduced High-Bandwidth Memory (HBM) [30]. The performance results using both memory subsystems follow the same trends; however for brevity and space constraints we only present the results using HBM as we believe it better captures the performance gains of our architectural change unhindered by the external memory subsystem which is not the focus of this work.

C. Experimental Results

Fig. 11(a) and 11(b) show the performance of the 8-bit and 4-bit DLA architecture respectively at different accelerator sizes using the original and modified FPGA. In the 8-bit DLA, the performance gap between the current and modified FPGAs increases as the accelerator size increases. With the original FPGA, programmable logic utilization increases rapidly to implement more multipliers after all the DSP blocks are used, leading to the steeper frequency decline with accelerator size as shown in Fig. 11(c) and hence lower performance. The scaling of both the BSC and ELT variations of DLA is limited by the available BRAMs although there is more room in the soft fabric for further scaling with ALM utilization of 77% and 58% in the original and modified FPGAs respectively. The DLA architecture suffers from this problem as the number of on-chip stream and weight buffers, required to supply the multipliers with sufficient bandwidth to perform a multiplication at every clock cycle, increases rapidly as we scale the number and length of the dot-product units inside each PE. However, the LRN variation was not able to scale to the largest size of the BSC and ELT variations in either the original and modified FPGA because of the heavily arithmetic NORM block that is completely implemented in the soft fabric. Similarly, the 4-bit DLA architectures showed increasing performance gains from the DSP modifications as the number of PEs is scaled up. Although the fracturable look-up table architecture with two bits of arithmetic in Intel FPGAs enables a highly efficient implementation of 4×4 multipliers in only 11 ALMs, our enhanced DSP block still resulted in approximately 1.5× performance gain on average. The scaling of all three variations in the 4-bit DLA on the modified FPGA is limited by the available on-chip BRAMs, while soft logic utilization is under 50%.

In the ASU architecture, we observe higher performance gains when using the proposed DSP blocks compared to those seen with the DLA architecture both in 8-bit and 4-bit precisions. As shown in Fig. 11(d) and 11(e), we could scale the architecture on the modified FPGA to much larger sizes than on the original FPGA as designs on it became unroutable at smaller sizes. In the 8-bit ASU architecture, even the first scaling from 3136 to 3920 multipliers degraded the performance on the original FPGA due to the very steep decline in frequency shown in Fig. 11(f) outweighing the increased number of PEs. The reason is that the ASU architecture has complex control logic and a data reuse register

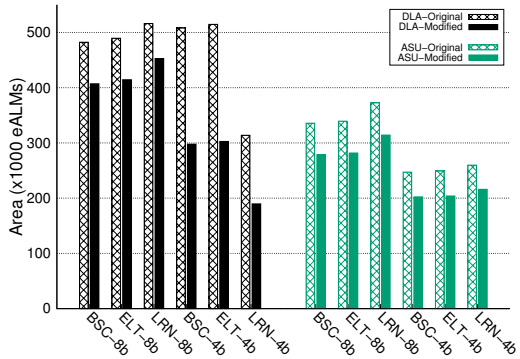


Fig. 12: Utilized area results for 8-bit/4-bit DLA and ASU on the original and modified FPGAs.

network which make it less routing-friendly compared to the modular pipelined daisy-chain architecture of DLA. When comparing the highest performing points on the original and modified FPGA for each variation of the benchmarks in Fig. 11(a), 11(b), 11(d) and 11(e), we obtain a 1.32× and 1.6× average performance enhancement in 8-bit and 4-bit benchmarks respectively from using the proposed DSP block. In the case where the multiplication operations are implemented only using DSP blocks (modeling scenarios in which the soft fabric is used to implement a different functionality), the performance gains of the enhanced DSP block are on average 1.62× and 2.97× for 8-bit and 4-bit benchmarks respectively.

Fig. 12 illustrates the utilized area of the largest accelerator size that could fit on the original FPGA compared to its area on the modified FPGA in equivalent ALMs (eALMs). In [31], the authors quantified the areas of an M20K BRAM and a Stratix-V DSP block (similar to our baseline DSP) to be 40× and 30× that of an ALM respectively. When measuring the utilized area of the benchmarks implemented on the modified FPGA, we consider the DSP block area to be 33.6 eALMs to account for the 12% block area increase. On average, our proposed DSP block resulted in 15% and 30% reduction in FPGA utilized area in the 8-bit and 4-bit benchmarks respectively, indicating that the enhanced DSP block can simultaneously improve performance and reduce the utilization of the FPGA's soft fabric. These area results also provide a first order approximation of the power reductions that can be achieved when using our enhanced DSP block.

VI. Conclusion

While a diversity of arithmetic precisions are used in deep learning inference tasks, the DSP blocks on current FPGAs lack the flexibility to efficiently implement the commonly used 8-bit and 4-bit precisions without underutilizing the DSP blocks. In this paper, we present a DSP block that can pack twice as many 9-bit and four times as many 4-bit multiplications as DSP blocks in current commercial FPGAs. We carefully study the trade-offs of circuitry reuse when adding the new modes to the DSP block to minimize the increase in block area. Adding support for four 9-bit and eight 4-bit independent multiplications and MAC modes resulted in 12% block area overhead which leads to an increase of only ∼0.6% in total core area of the FPGA. Finally, we quantify the performance gains of using our proposed DSP block in two state-of-the-art convolutional neural network accelerators, Intel's DLA and ASU architectures, on a variety of models (AlexNet, VGG-16 and ResNet-50) at both 8-bit and 4-bit precisions. On average, an FPGA incorporating our enhanced DSP block not only achieves 1.32× and 1.6× higher performance, but also has 15% and 30% less utilized area for 8-bit and 4-bit benchmarks respectively.

Acknowledgments

We would like to thank Jonathan Greene from Microsemi for insightful discussions, and the NSERC/Intel industrial research chair in programmable silicon and Huawei for funding support.

References

[1] S. Han *et al.*, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," *NIPS*, 2015.
[2] P. Gysel *et al.*, "Hardware-oriented Approximation of Convolutional Neural Networks," *ICLR*, 2016.
[3] S. Han *et al.*, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *ISCA*, 2016.
[4] A. Zhou *et al.*, "Incremental Network Quantization: Towards Lossless CNNs with Low-precision Weights," *arXiv:1702.03044*, 2017.
[5] A. Mishra *et al.*, "WRPN: Wide Reduced-Precision Networks," *arXiv:1709.01134*, 2017.
[6] Nvidia Corporation, "Nvidia Tesla P40 Inferencing Accelerator," 2016.
[7] N. P. Jouppi *et al.*, "In-datacenter Performance Analysis of a Tensor Processing Unit," in *ISCA*, 2017.
[8] Y. Fu *et al.*, "Deep Learning with INT8 Optimization on Xilinx Devices," *white paper of Xilinx*, 2016.
[9] J. L. Holt and T. E. Baker, "Back Propagation Simulations using Limited Precision Calculations," in *IJCNN*, vol. 2, 1991.
[10] S. Gupta *et al.*, "Deep Learning with Limited Numerical Precision," in *ICML*, 2015.
[11] M. Courbariaux *et al.*, "Training Deep Neural Networks with Low Precision Multiplications," *arXiv:1412.7024*, 2014.
[12] K. Hwang and W. Sung, "Fixed-point Feedforward Deep Neural Network Design using Weights+ 1, 0, and- 1," in *SiPS*, 2014.
[13] N. Mellempudi *et al.*, "Mixed Low-precision Deep Learning Inference using Dynamic Fixed Point," *arXiv:1701.08978*, 2017.
[14] E. Chung and J. Fowers, "Accelerating Persistent Neural Networks at Datacenter Scale," in *HOT CHIPS*, vol. 29, 2017.
[15] Intel Corporation, "Intel Stratix 10 Variable Precision DSP Blocks User Guide (UG-S10-DSP)," 2017.
[16] M. Langhammer and B. Pasca, "Floating-point DSP Block Architecture for FPGAs," in *FPGA*, 2015.
[17] Xilinx Inc., "Ultrascale Architecture DSP Slice (UG579)," 2017.
[18] Microsemi Corporation, "Efficient INT8 Dot Product using Microsemi Math Block (WP0216)," 2018.
[19] H. P. Afshar and P. Ienne, "Highly Versatile DSP Blocks for Improved FPGA Arithmetic Performance," in *FCCM*, 2010.
[20] R. Warrier *et al.*, "Fracturable DSP Block for Multi-context Reconfigurable Architectures," *CSSP*, vol. 36, no. 7, 2017.
[21] C. R. Baugh and B. A. Wooley, "A Two's Complement Parallel Array Multiplication Algorithm," *TC*, vol. 100, no. 12, 1973.
[22] L. Dadda, "Some Schemes for Parallel Multipliers," *Alta frequenza*, vol. 34, no. 5, 1965.
[23] M. Langhammer, "Digital Signal Processing Circuitry with Redundancy and Ability to Support Larger Multipliers," 2014, US Patent 8,886,696.
[24] Intel Corporation, "Stratix V Device Overview (SV51001)," 2015.
[25] S. Yazdanshenas and V. Betz, "Automatic Circuit Design and Modelling for Heterogeneous FPGAs," in *FPT*, 2017.
[26] H. Wong *et al.*, "Quantifying the Gap between FPGA and Custom CMOS to Aid Microarchitectural Design," *TVLSI*, vol. 22, no. 10, 2014.
[27] U. Aydonat *et al.*, "An OpenCL Deep Learning Accelerator on Arria 10," in *FPGA*, 2017.
[28] Y. Ma *et al.*, "Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks," in *FPGA*, 2017.
[29] P. Colangelo *et al.*, "Exploration of Low Numeric Precision Deep Learning Inference using Intel FPGAs," in *FCCM*, 2018.
[30] Xilinx Inc., "Virtex Ultrascale+ HBM FPGA: A Revolutionary Increase in Memory Performance (WP485)," 2017.
[31] R. Rashid *et al.*, "Comparing Performance, Productivity and Scalability of the TILT Overlay Processor to OpenCL HLS," in *FPT*, 2014.