WP490 (v1.0.1) April 19, 2017

Embedded Vision with INT8 Optimization on Xilinx Devices

By: Yao Fu, Ephrem Wu, Varun Santhaseelan, Kristof Denolf, Kamran Khan, and Vinod Kathail

Xilinx INT8 optimization provides the best performance and most power efficient computational techniques for embedded vision applications using deep learning inference and traditional computer vision functions. Xilinx's integrated DSP architecture can achieve 1.75X greater solution-level performance with INT8 operations than other FPGA DSP architectures.

ABSTRACT

This white paper explores INT8 operations for embedded vision applications using deep learning inference and computer vision functions implemented on the Xilinx DSP48E2 slice, and how this contrasts with other FPGAs. Xilinx's DSP architecture can achieve 1.75X peak solution-level performance for INT8 multiply and accumulate (MACC) operations compared to other FPGAs with the same resource count. Embedded vision applications leverage lower bit precision without sacrificing accuracy, so efficient INT8 implementations are needed.

Xilinx's DSP architecture and libraries are optimized for INT8 operations. This white paper describes how the DSP48E2 slice in Xilinx's 16nm and 20nm All Programmable devices can be used to process two concurrent INT8 MACC operations while sharing the same kernel weights. It also explains why 24-bit is the minimal size for an input to utilize this technique, which is unique to Xilinx. Additionally, this white paper details how the DSP48E2 slice can be used in SIMD mode for basic arithmetic operations. Examples of how these capabilities can be leveraged for embedded vision, in the context of deep learning or other computer vision processing tasks, are provided.

INT8 for Deep Learning and Computer Vision

Embedded vision is the term coined for implementing computer vision algorithms for real-world purposes on embedded platforms. While computer vision algorithms have improved significantly in recent years, the challenge of porting such complex and compute-intensive algorithms to embedded platforms with less power consumption is a major challenge. There is a constant need to process more operations using less power, whether it is for traditional computer vision algorithms, like filtering, corner detection, etc., or for deep learning algorithms.

Deep neural networks have propelled an evolution and redefined many applications with human-level AI capabilities. These networks are a key workload in embedded devices, given the superior accuracy provided by the algorithms. While more accurate deep learning models have been developed, their complexity is accompanied by high compute and memory bandwidth challenges. Power efficiency is driving innovation in developing new deep learning inference models that require lower compute intensity and memory bandwidth, but this must not be at the cost of accuracy and throughput. Reducing the overhead ultimately increases power efficiency and lowers the total power required.

In addition to saving power during computation, lower bit-width compute also lowers the power needed for memory bandwidth, because fewer bits are transferred with the same amount of memory transactions.

Research has shown that floating point computations are not required in deep learning inferences to keep the same level of accuracy [Ref 1][Ref 2][Ref 3], and many applications, such as image classification, only require INT8 or lower fixed point compute precision to keep an acceptable inference accuracy [Ref 2][Ref 3]. Table 1 shows fine-tuned networks with dynamic fixed point parameters and outputs for convolutional and fully connected layers. The numbers in parentheses indicate accuracy without fine-tuning.

Table 1: **CNN Models with Fixed-Point Precision**

	Layer Outputs	CONV Parameters	FC Parameters	32-Bit Floating Point Baseline	Fixed Point Accuracy
LeNet (Exp1)	4-bit	4-bit	4-bit	99.1%	99.0% (98.7%)
LeNet (Exp2)	4-bit	2-bit	2-bit	99.1%	98.8% (98.0%)
Full CIFAR-10	8-bit	8-bit	8-bit	81.7%	81.4% (80.6%)
SqueezeNet top-1	8-bit	8-bit	8-bit	57.7%	57.1% (55.2%)
CaffeNet top-1	8-bit	8-bit	8-bit	56.9%	56.0% (55.8%)
GoogLeNet top-1	8-bit	8-bit	8-bit	68.9%	66.6% (66.1%)

The optimization of INT8 operations for deep learning is also directly applicable to a large set of traditional computer vision functions. These algorithms typically operate on 8- to 16-bit integer representations. OpenVX[Ref 4], a recently proposed standard for computer vision, specifies the use of INT8 representation per channel. Most computer vision applications require some level of filtering, which can be decomposed into a set of dot product operations. The SIMD mode of operation on a Xilinx DSP48E2 slice provides additional options to implement operations involved in vision algorithms.

INT8 Operations on Xilinx DSP Slices

Xilinx's DSP48E2 slice, in UltraScale and UltraScale+ FPGAs, and Zynq UltraScale+ MPSoCs (Programmable Logic), is designed to do one multiplication and addition operation, with up to 18x27 bit multiplication and up to 48-bits accumulation, efficiently within one clock cycle as shown in Figure 1. While looping back to itself or chaining multiple DSP48E2 slices together, multiplication and accumulation (MACC) can also be done efficiently with Xilinx devices.

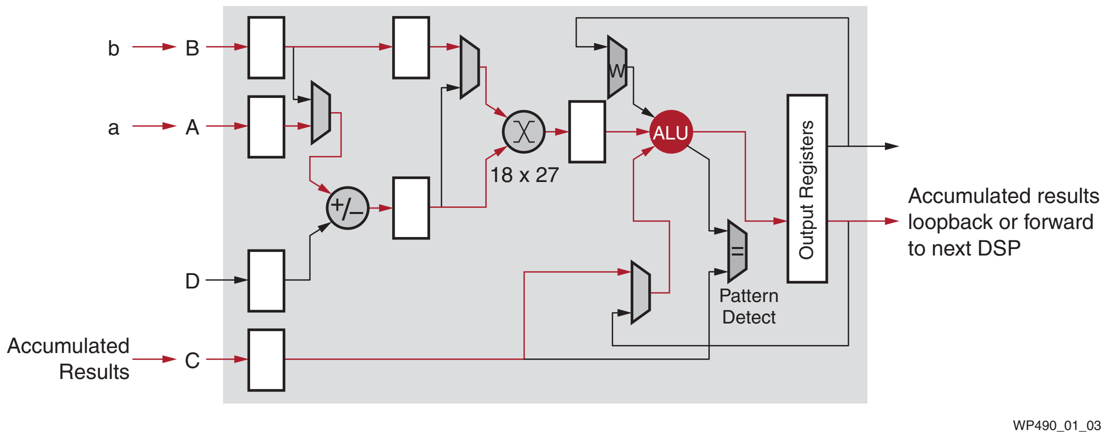

Figure 1: **DSP48E2 Slice with MACC Mode**

INT8 computations innately take advantage of the wide 27-bit width. In traditional applications, the pre-adder is usually utilized to implement (A+B) x C type of computations efficiently, but this type of computation is not very often seen in deep learning and computer vision applications. Separating the result of (A+B) x C into A x C and B x C enables the accumulation in a separate dataflow, allowing it to fit a typical computation relevant in deep learning and computer vision.

Having an 18x27 bit multiplier is an advantage for INT8 MACC operations. At a minimum, one of the inputs to the multiplier needs to be at least 24 bits and the carry accumulator needs to be 32 bits to perform two INT8 MACCs concurrently on one DSP48E2 slice. The 27-bit input can be combined with a 48-bit accumulator to achieve a 1.75X solution performance improvement (1.75:1 DSP multiplier to INT8 MACC ratio). FPGAs from other vendors only have an 18x19 multiplier in a single DSP block and are limited to a 1:1 ratio of DSP multiplier to INT8 MACC.

Scalable INT8 Optimization

The goal is to find a way to efficiently encode input *a*, *b*, and *c* so that the multiplication results between *a*, *b,* and *c* can be easily separated into *a x c* and *b x c*. Given the common input c, the technique is referred to as a single instruction, 2 data with common coefficient.

In a reduced precision computation, e.g., INT8 multiplication, the higher 10-bit or 19-bit inputs are filled with `0`s or `1`s, and carry only one bit of information. This is the same for the upper 29-bits of the final 45-bit product. Consequently, it is possible to use the higher 19 bits to carry another computation while the lower 8-bit and 16-bit input results are not affected.

Generally, two rules must be followed to utilize the unused upper bits for another computation:

1. Upper bits should not affect the computation of the lower bits.
2. Any contamination of the upper bits by the computation of the lower bits must be detectable and recoverable.

To satisfy the above rules, the least significant bit of the upper product results must not fall into the lower 16 bits. Thus, the upper bits' input should start with at least the 17th bit. For an 8-bit upper input that requires a minimum of 16 + 8 = 24-bits total input size. This minimum 24-bit input size can only guarantee two concurrent multiplications with one multiplier—but still not enough to reach the overall 1.75X MACC throughput.

Following are the steps to compute *ac* and *bc* in parallel in one DSP48E2 slice, which is used as an arithmetic unit with a 27-bit pre-adder (both inputs and outputs are 27 bits wide) and a 27x18 multiplier. See Figure 2.

1. Pack 8-bit input *a* and *b* in the 27-bit port *p* of the DSP48E2 multiplier via the pre-adder so that the 2-bit vectors are as far apart as possible.
   The input *a* is left-shifted by only 18 bits so that two sign bits *a* in the 27-bit result from the first term to prevent overflow in the pre-adder when *b*<0 and *a* = –128. The shift amount for *a* being 18, or the width of the DSP48E2 multiplier port B, is coincidental.

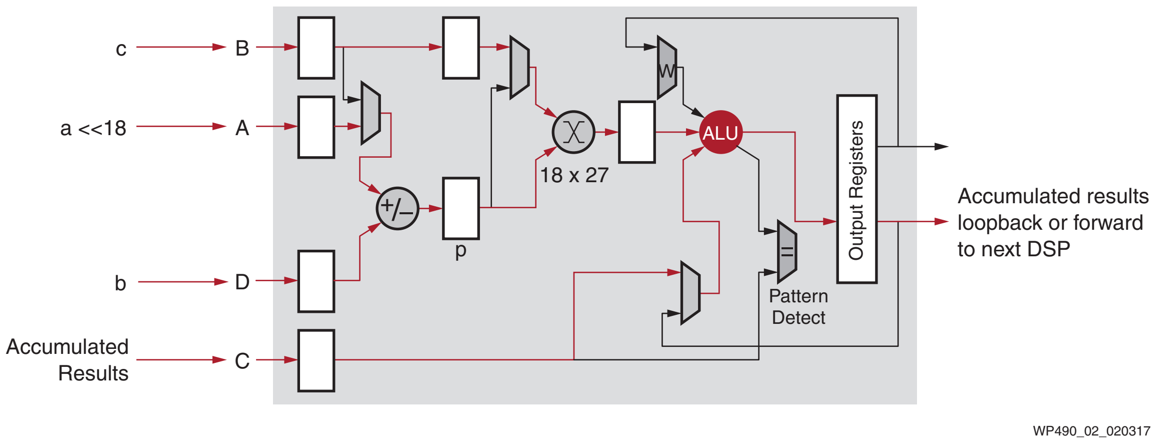

Figure 2: **8-Bit Optimization**

2. The DSP48E2 27x18 multiplier is used to compute the product of the packed 27-bit port *p* and an 8-bit coefficient represented in 18-bit *c* in twos complement format. This 45-bit product is the sum of two 44-bit terms in twos complement format: *ac* left-shifted by 18 bits, and *bc*.

The post adder can be used to accumulate the above 45-bit product, which contains separable upper and lower product terms. Correct accumulations are carried for the upper and lower terms while accumulating the single 45-bit product. The final accumulation results, if not overflowed, can be separated by simple operations.

The limitation of this technique is the number of product terms each DSP48E2 slice can accumulate. With 2 bits remaining between the lower and upper product terms (Figure 3), accumulation of up to seven product terms only can be guaranteed with no overflow for the lower bits. After seven product terms, an additional DSP48E2 slice is required to extend this limitation. As a result, these eight DSP48E2 slices perform 7x2 INT8 multiply-add operations, 1.75X the INT8 MACC operations compared to competitive devices with the same number of multipliers.

There are many variations of this technique, depending on the requirements of actual use cases. Convolutional neural networks (CNN) with rectified linear unit (ReLU) produce non-negative activation, and the *unsigned* INT8 format creates one more bit of precision and 1.78X peak throughput improvement.

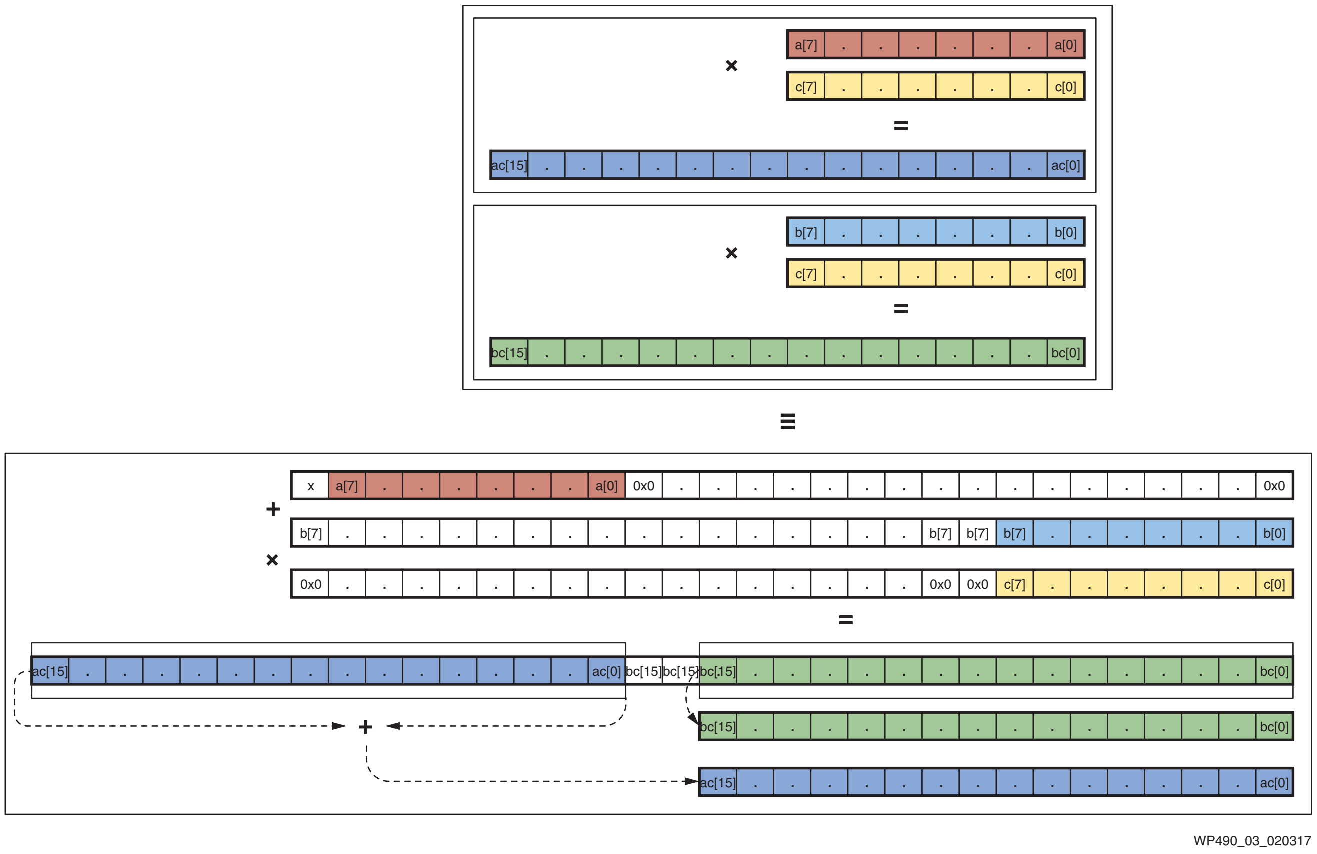

Figure 3: **Packing Two INT8 Multiplication with a Single DSP48E2 Slice**

DSP48E2 SIMD Mode

The post-adder of the DSP48E2 slice is fractured into four 12-bit or two 24-bit SIMD ALUs (see Figure 4) to perform parallel addition, subtraction, accumulation, or bit-wise logic operations. In the SIMD mode, the pre-adder and the multiplier of the DSP48E2 slice are unavailable. On a per-cycle basis, the ALUMODE[3:0] control bus selects the operation while the OPMODE[8:0] control bus selects the operands W, X, Y, and Z. If considering 24-bit operations, the P register of the DSP48E2 slice can store the result of processing two input arrays. For each array, summing is performed serially, one element per cycle. The throughput is thus two new results per cycle. Details are provided in UG579, *UltraScale Architecture DSP Slice User Guide* (keywords "SIMD," "ALUMODE," and "OPMODE") [Ref 5].

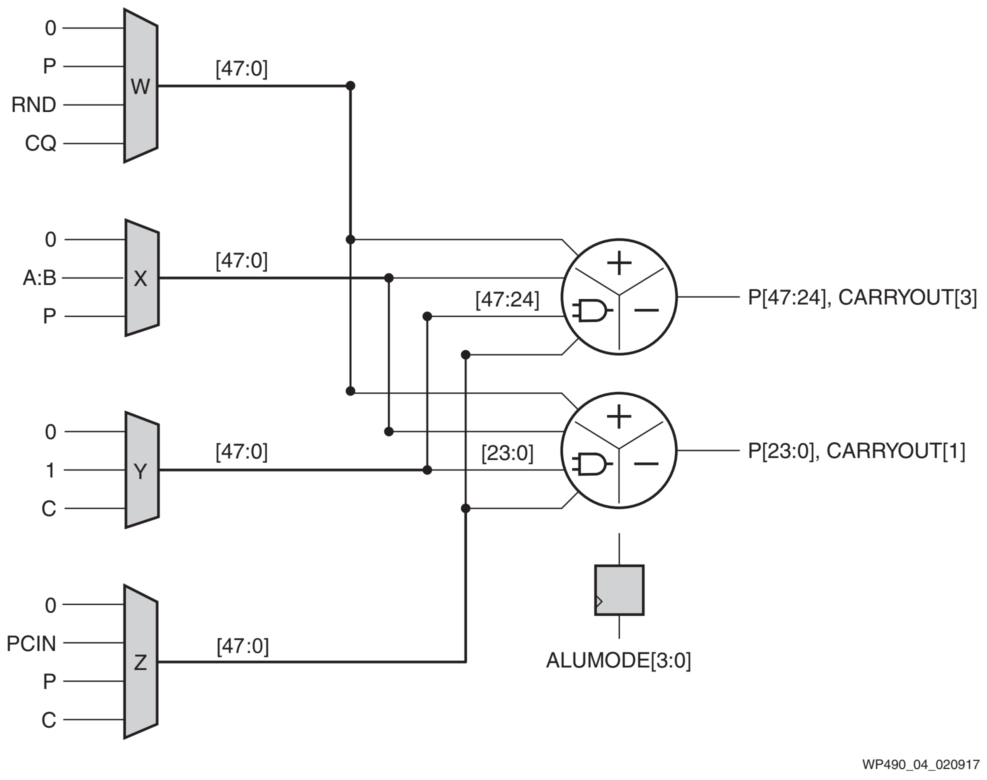

Figure 4: **DSP48E2 Dual 24-Bit SIMD Mode**

Mapping INT8 Optimizations to Deep Learning Applications

Modern neural networks are mostly derived from the original perceptron model [Ref 6]. See Figure 5.

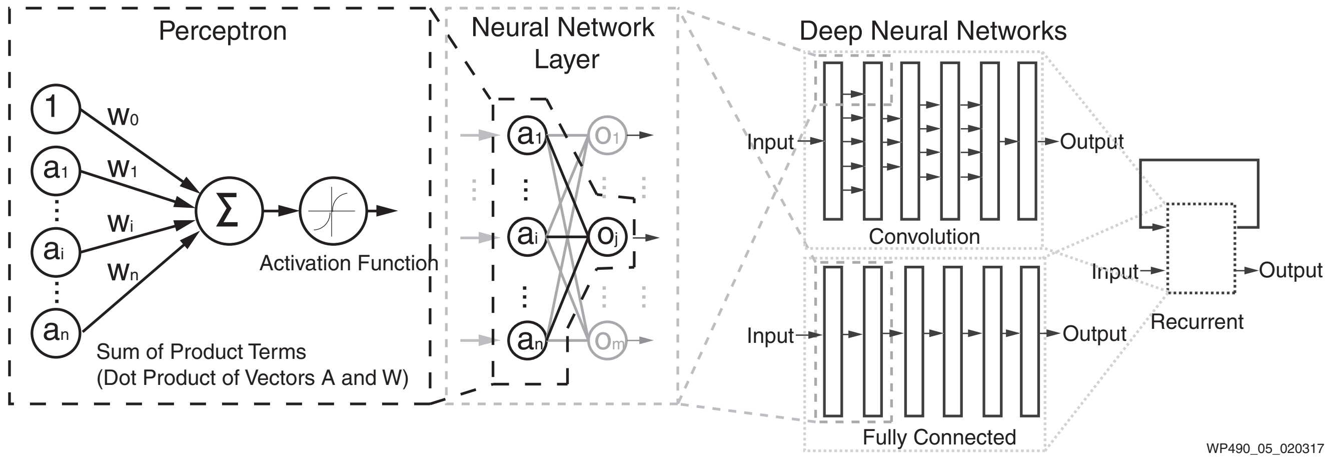

Figure 5: **Perceptron and Deep Neural Networks**

Although quite evolved from the standard perceptron structure, the basic operations of modern deep learning, also known as deep neural networks (DNN), are still perceptron-like operations, but in wider ensemble and deeper stacked perceptron structures. Figure 5 shows the basic operation of a perceptron, through multiple layers, and ultimately repeated millions to billions of times in each typical deep learning inference. As shown in Figure 6, the major compute operations for computing each of the *m* perceptron/neuron outputs

$$o_j \ (j \in [1, m])$$

in a layer of neural networks is: to take the entire *n* input samples

$$a_i (i \in [1, n])$$

multiply each input by the corresponding kernel weight

$$w_{i,j} \ (i \in [1, n], j \ \in [1, m])$$

and accumulate the results

$$o_j = f(\sum_i a_i \, w_{i,j}) \, , \qquad (i \in [1, n])$$

Where ***f(x)*** can be any activation function of choice.

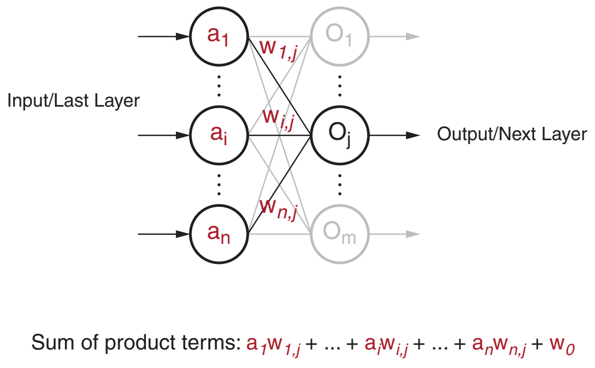

Figure 6: **Perceptron in Deep Learning**

If the precision of a_i and $w_{i,j}$ are limited to INT8, this sum of products is the first of the parallel MACCs described in the INT8 optimization technique.

The second sum of the product uses the same input $a_i \ (i \in [1, n])$, but a different set of kernel weights $w_{i,k} \ (i \in [1, n], k \ \in [1, m], and \ k \ \neq j)$

The result of the second perceptron/neuron output is

$$o_k = f(\sum_i a_i \, w_{i,k}) \, , \qquad (i \in [1, n], k \ \neq j)$$

See Figure 7.

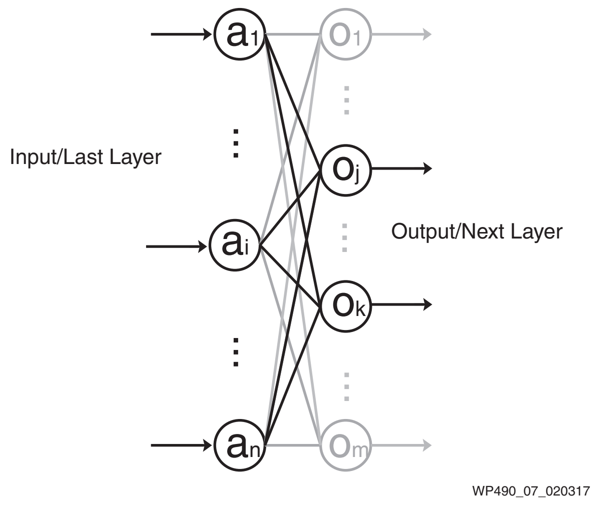

Figure 7: **Two Sums of Product Terms in Parallel with Shared Input**

By shifting the $w_{i,k}$ values 18 bits to the left using the INT8 optimization technique, each DSP48E2 slice provides a partial and independent portion of the final output values. The accumulator for each of the DSP48E2 slices is 48 bits wide, and is chained to the next slice. This limits the number of chained blocks to seven, before saturation of the shifted $w_{i,k}$ affects the calculation, i.e., 2*n* MACCs with *n* DSP slices for the total of *n* input samples.

Each layer of a typical DNN has 100s to 1000s of input samples. However, after seven terms of accumulation, the lower terms of the 48-bit accumulator might be saturated, and an extra DSP48E2 slice is needed for the summation every seven terms. This equates to 14 MACCs with every 7 DSP48E2 slices plus 1 DSP48E2 slice for preventing the oversaturation, resulting in a throughout improvement of 7/4 or 1.75X.

In convolution neural networks (CNN), the same set of weights is usually reused heavily in convolutional layers, thus form *a x w*, and *b x w* type of parallel MACCs operations. So weight sharing instead of input sharing can also be used (see Figure 8).

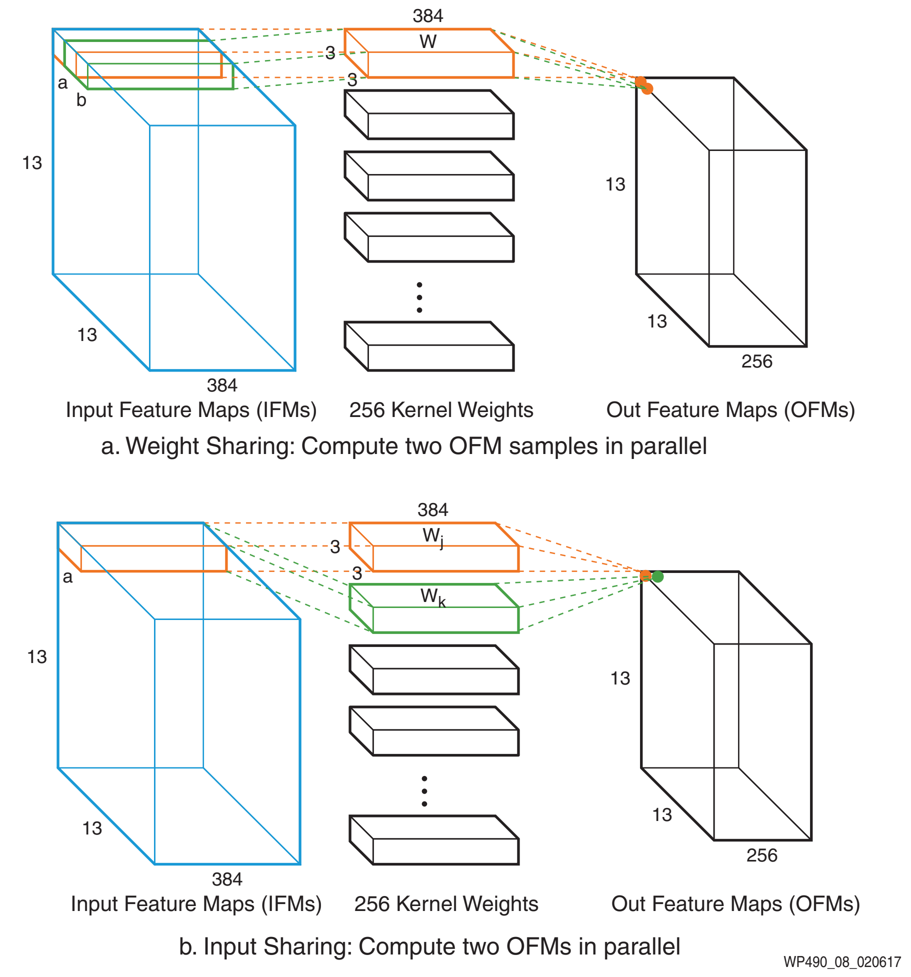
384
W
3
3
a
b
13
13
384
13
13
256
Input Feature Maps (IFMs)
256 Kernel Weights
Out Feature Maps (OFMs)
a. Weight Sharing: Compute two OFM samples in parallel
384
W_j
3
3
W_k
a
13
13
384
13
13
256
Input Feature Maps (IFMs)
256 Kernel Weights
Out Feature Maps (OFMs)
b. Input Sharing: Compute two OFMs in parallel
WP490_08_020617

Figure 8: **Weight Sharing and Input Sharing Comparison**

Other Methods to Create INT8 Chained MACCs

INT8 chained MACCs can also be constructed using available LUTs, i.e., LUTs not used in other parts of the design, in the programmable logic that operate at a similar frequency to the DSP48E2 slice.

Leveraging available LUTs can substantially increase the deep learning performance, in some cases by 3X. In many instances, with respect to other non-FPGA architectures, these available compute resources are not accounted for when calculating the available deep learning operations.

The Programmable Logic in Xilinx FPGAs and MPSoCs is unique because it can handle diverse workloads concurrently and efficiently. For example, Xilinx FPGAs and MPSoCs can perform CNN image classification, networking cryptography, and data compression concurrently. This deep-learning performance competitive analysis does not take the MACC LUTs into account because LUTs are usually more valuable while being used to perform other concurrent functions rather than to perform MACC functions.

Mapping INT8 Optimizations to Computer Vision Functions

The Khronos OpenVX standard defines a set of computer vision processing modules that are especially important in use cases such as: face, body, and gesture tracking; smart video surveillance; advanced driver assistance systems (ADAS); object and scene reconstruction; augmented reality; visual inspection; robotics; and more. Table 2 shows computer vision related functions for which INT8 optimization is applicable.

Table 2: **Applicable INT8 Optimizations for Computer Vision Functions**

Image / Vision Processing Benchmark Components	OpenVX	OpenCV	Scalable INT8 Compatible	SIMD Compatible
Pixel-level: Spatial Filtering and Geometric Spatial Transforms				
Absolute Difference	AbsDiff	absDiff	–	Y
Accumulate Image	AccumulateImage	accumulate	–	Y
Accumulate Weighted	AccumulateWeightedImage	accumulateWeighted	Y	–
Arithmetic Add	Add	add	–	Y
Arithmetic Substract	Substract	subtract	–	Y
Bitwise And, Or, Xor, Not	And, Or, Xor, Not	bitwise_and, bitwise_or, bitwise_xor, bitwise_not	–	Y
Threshold	Threshold	threshold	–	Y
Box Filter	Box3x3	boxFilter	Y	–
Filter 2D (Convolution)	Convolve	filter2D	Y	–
Dilation Filter (Morphological), Max	Dilate3x3	dilate	–	Y
Erosion Filter (Morphological), Min	Erode3x3	erode	–	Y
Gaussian Filter	Gaussian3x3	GaussianBlur	Y	–
Gaussian Pyramid	GaussianPyramid	buildPyramid	Y	–
Laplacian Pyramid	LaplacianPyramid		Y	–
Median	Median3x3	medianBlur		Y
Sobel	Sobel3x3	Sobel	Y	–
Scaling	HalfScaleGaussian, ScaleImage	resize	Y	–
Optical Flow	OpticalFlowPyrLK	PyrLKOpticalFlow	Y	–
Stereo Disparity (SBM or SGM)	N\A	FindStereoCorrespondence BM	Y	–
Analysis and Recognition				
Integral Image	IntegralImage	integral	–	Y
Mean Std Deviation	MeanStdDev	meanStddev	–	Y
Min Max Locations	MinMaxLoc	minMaxLoc	–	Y
Canny Edge Detection	CannyEdgeDetector	Canny	Y	Y

Table 2: **Applicable INT8 Optimizations for Computer Vision Functions *(Cont'd)***

Image / Vision Processing Benchmark Components	OpenVX	OpenCV	Scalable INT8 Compatible	SIMD Compatible
FAST9 Corner Detection	FastCorners	FAST	Y	–
Harris Corner Detection	HarrisCorners	cornerHarris	Y	–
Oriented FAST and Rotated BRIEF Feature Detector	N\A	ORB::ORB	–	Y
HOG Detection / Descriptors	N\A	HOGDescriptor	Y	–
SVM	N\A	CvSVM::predict	Y	–

Scalable INT8 optimization checks the compatibility to process two data items at the same time with a shared coefficient. SIMD checks those modules that can benefit from the four operand operators in the DSP48E2 slice. All filter-related modules where data and weights respect the 8-bit limitation benefit from the scalable INT8 technique. Most other modules involving basic image arithmetic (like add/subtract or compare, etc.) can leverage the DSP48E2's SIMD operations.

Custom 2D Convolution Using Scalable INT8 Optimization

In the context of computer vision functions, most pre-processing tasks involve some level of filtering. Since images are mostly represented using 8 bits per channel, the optimization for INT8 operations in deep learning applications can be applied to two dimensional filtering operations in image processing. The only constraint is that the coefficients in the filter must have precision that can be represented in 8 bits. This is generally true in cases of common filters like Sobel, Scharr, Laplacian, or other edge detection filters.

The two-multiplier mode in a DSP48E2 slice can be leveraged using one of the following techniques:

Operate on multiple pixels of output for the same channel: In this mode, two output pixels can be operated on in parallel. Since coefficients of the filter are shared between pixels in the image, the pixel at location (x,y) and (x,y+1) can be computed concurrently. Each filter coefficient is multiplied with two different input pixels depending on the order in which the filter computation occurs. This means that performance increases by 1.75X given the same amount of resources available in the Programmable Logic.

Operate on multiple pixels of output for different channels or images: If the image being processed has multiple channels and the filter is shared between different channels, the coefficient of filter can be shared across channels for the pixel at same location at (x,y). The same technique could be extended to operate on multiple images at the same time.

Median Filter Using SIMD Operation

The median filter, also popular in image processing, is used for noise removal. Using the median filter on an image involves scanning the image with a window of a preset size, computing the median of pixels that fall within that window, and replacing the center pixel with the median value. Calculating the median is compute-intensive. It involves sorting the values and then finding the value that falls in the middle of the list. The sorting procedure is a sequence of comparison operations.

To implement a median filter using DSPs on the Programmable Logic, the algorithm can be modified. Each comparison operation can be split into a subtract operation followed by checking the sign-bit. The DSP48E2 slice is capable of operating in quad 12-bit or dual 24-bit mode for a subtraction operation. To fully utilize the DSP48E2 slice, it is possible to operate on multiple pixels in parallel. Assuming each pixel has a single channel with a depth less than 12 bits, it is possible to process four output pixels concurrently. For each output pixel, there are multiple sorting operations for which the subtract operation in the DSP48E2 slice is used. The sign bit of the result can be checked outside of the DSP48E2 slice with minimal logic. The total number of comparisons depends on the algorithm used to sort the values.

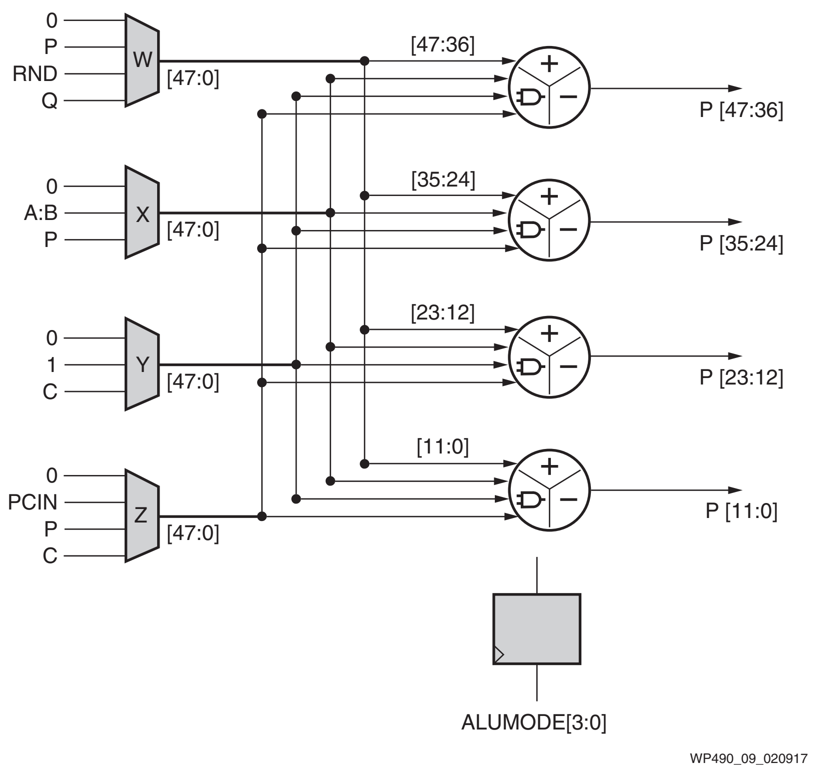

Figure 9: **DSP48E2 Operating Mode for Median Filter**

Competitive Analysis

Intel's (formerly Altera) Arria 10 devices are used in this competitive analysis against Xilinx's Zynq® UltraScale+™ MPSoC. For this compute efficiency comparison in embedded vision applications, the devices chosen have comparable DSP density and device power:

- Arria 10 SoCs: SX220, SX270, and SX480
- Zynq UltraScale+ MPSoCs: ZU3, ZU7, and ZU9 devices

This comparison focuses on general-purpose MACC performance that can be used in many applications, such as deep learning and computer vision.

Intel's MACC performance is based on operators that leverage the pre-adders. However, this implementation produces the sum of product terms and not unique separate product terms. As a result, Intel's pre-adders are not suited for efficient deep learning or computer vision operations.

For this compute efficiency analysis, the power for each of the devices is estimated using Xilinx's 2016.4 Power Estimator tool and Intel's 16.0.1 EPE Power Estimate tools with the following assumptions:

1. 90% DSP utilization
2. Intel devices -2L, 0.9V at F_{MAX}
3. Xilinx devices -1L, 0.72V at F_{MAX}
4. 70% logic utilization with clock rate at DSP F_{MAX}
5. 90% block RAM utilization with the clock rate at half DSP F_{MAX}
6. 12.5% DSP toggle rate
7. Power characteristics: “Typical Power”

Figure 10 shows the power efficiency comparison of deep learning and compute vision operations. Compared to Intel's Arria 10 SoCs devices, Xilinx devices deliver 3–7X higher compute efficiency on deep learning and computer vision operations.

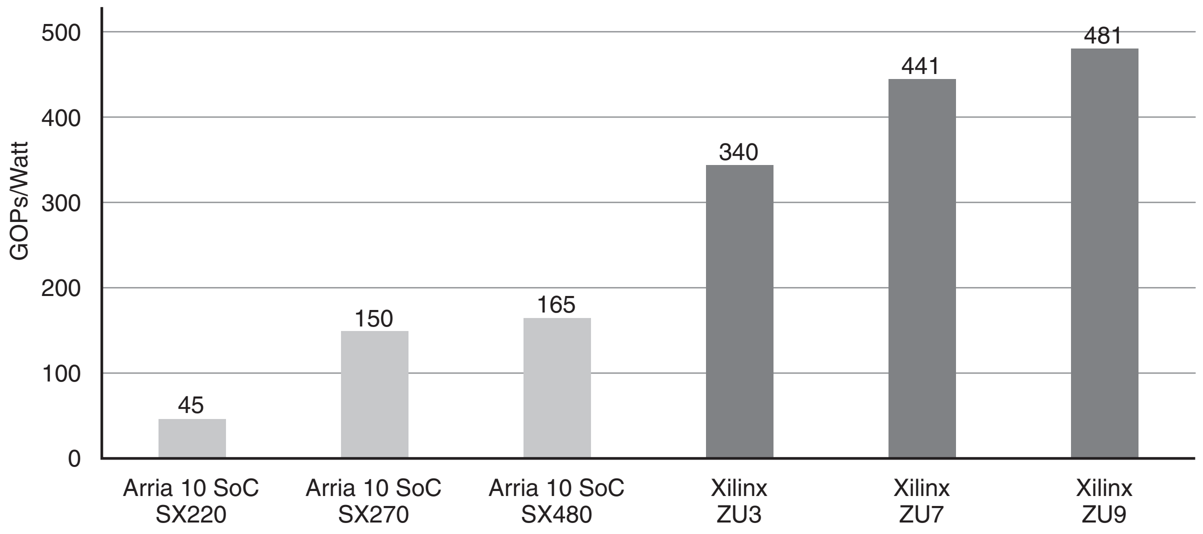

Figure 10: **INT8 Deep Learning and Computer Vision Power Efficiency Comparison: Xilinx vs. Intel**

Conclusion

This white paper explores how INT8 deep learning operations and computer vision are optimal on Xilinx DSP48E2 slices, achieving a 1.75X performance gain. For deep learning, the Xilinx DSP48E2 slice can be used to do two concurrent INT8 MACCs while sharing the same kernel weights. To implement INT8 efficiently, an input width of 24 bits is required, an advantage that is only supported in Xilinx's DSP48E2 slices. The same advantage can be exploited in computer vision for operations like filtering and other image manipulation tasks. Xilinx's DSP48E2 slice's SIMD mode provides additional means to perform four 12-bit or two 24-bit SIMD operations.

In summary, Xilinx's Zynq UltraScale+ MPSoC is well suited for INT8 workloads, which makes it a perfect choice for accelerating a large number of applications in the embedded vision area. Xilinx is continuing to innovate new hardware- and software-based methodologies to accelerate deep learning and computer vision functions in embedded vision applications.

For more information on embedded vision on Xilinx devices, go to: www.xilinx.com/revision

References

1. Dettmers, 8-Bit Approximations for Parallelism in Deep Learning, ICLR 2016
https://arxiv.org/pdf/1511.04561.pdf
2. Gysel et al, Hardware-oriented Approximation of Convolutional Neural Networks, ICLR 2016
https://arxiv.org/pdf/1604.03168v3.pdf
3. Han et al, Deep Compression: Compressing Deep Neural Networks With Pruning, Trained Quantization And Huffman Coding, ICLR 2016
https://arxiv.org/pdf/1510.00149v5.pdf
4. Khronos Group, https://www.khronos.org/openvx/
5. UG579, *UltraScale Architecture DSP Slice User Guide*
6. Rosenblatt, F., The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, Psychological Review, Vol. 65, No. 6, 1958
http://www.ling.upenn.edu/courses/cogs501/Rosenblatt1958.pdf

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
04/19/2017	1.0.1	Typographical edit.
03/10/2017	1.0	Initial Xilinx release.

Disclaimer

Automotive Applications Disclaimer