WP521 (v1.0.1) June 24, 2020

Convolutional Neural Network with INT4 Optimization on Xilinx Devices

INT8 provides better performance with comparable precision than floating-point for AI inference. But when INT8 is unable to meet the desired performance with limited resources, INT4 optimization is the answer. With INT4 optimization, Xilinx can achieve up to a 77% performance boost on real hardware in comparison with the current INT8 solution.

ABSTRACT

Xilinx provides an INT8 AI inference accelerator on Xilinx hardware platforms—Deep Learning Processor Unit (XDPU). However, in some resource-limited, high-performance and low-latency scenarios (such as the resource-power-sensitive edge side and low-latency ADAS scenario), low bit quantization of neural networks is required to achieve lower power consumption and higher performance than provided by INT8. However, extremely low bit quantization (such as binary or ternary) has accuracy degradation.

Thus, a full-process hardware-friendly quantization solution of 4-bit activations and 4-bit weights (4A4W) achieves better accuracy/resource trade-off. This white paper describes the implementation of a low-precision accelerator for CNN 4-bit XDPU on the Zynq® UltraScale+™ MPSoC and Zynq-7000 SoC families (16nm and 28nm), which takes full advantage of its DSP capabilities by efficiently mapping convolutional computations. This solution achieves 2X solution-level performance over the XDPU. On a 2D detection task in an ADAS system, the implementation achieves an inference speed of 230fps on a Zynq UltraScale+ MPSoC ZCU102 board, which is a 1.52X performance gain over the 8-bit XDPU. In addition, this solution achieves comparable results to full-precision models on different tasks of the ADAS system.

Introduction

Companies are increasingly focused on productization of AI-based systems in fields such as data center, automotive, industrial, and medical. This presents two major challenges:

- AI inference requires orders of magnitude more computation, while keeping the price, power, latency, and form factor intact.
- AI scientists continue to innovate daily on the algorithms and models that require different hardware architecture to be optimal.

The high demand for constant innovation requires an adaptable Domain Specific Architecture (DSA). One of the key trends in optimizing AI inference performance and reducing the power is to use lower and mixed precisions. To reduce hardware design complexity, model quantization is applied as a key technology to various hardware platforms. A lot of work has been devoted to minimizing CNN computation and storage costs. This work extensively demonstrates that weights and activations can be represented by INT8 without a significant decrease in accuracy for most computer vision tasks. But hardware resources are still insufficient for some edge applications. When a lower bit width (e.g., 1 bit, 2 bits) is used for edge applications, some popular hardware design solutions use the simplified multiplier. Although these solutions can get low latency and high throughput, they still cause a large accuracy gap on a full-precision model. Therefore, it is necessary to find a balance between model accuracy and hardware performance.

Xilinx experimented with using different quantization methods on ImageNet classification [Ref 1] tasks with several popular network structures((ResNet50V1 [Ref 2], ResNet50V2 [Ref 3], MobilenetV1 [Ref 4], MobilenetV2 [Ref 5]). The results showed that the accuracy decreases as the bit width decreases. Especially when the bit width is less than 4, the accuracy drop is significant. Xilinx also used a roofline model, as introduced by Williams et al. [Ref 6], to analyze hardware performance with different bit-widths, as shown in Figure 1. Using a Xilinx ZCU102 evaluation board as an example, as the precision of the MAC decreases, the hardware cost decreases and the performance increases. The experiment also showed that low bit quantization can improve performance by reducing the memory requirement. This is illustrated by the operational intensity of convolution in the ResNet-50 neural network, which is shown for the two cases of the 8-bit precision and 4-bit precision operation. Therefore, INT4 is the best trade-off between model accuracy and hardware performance.

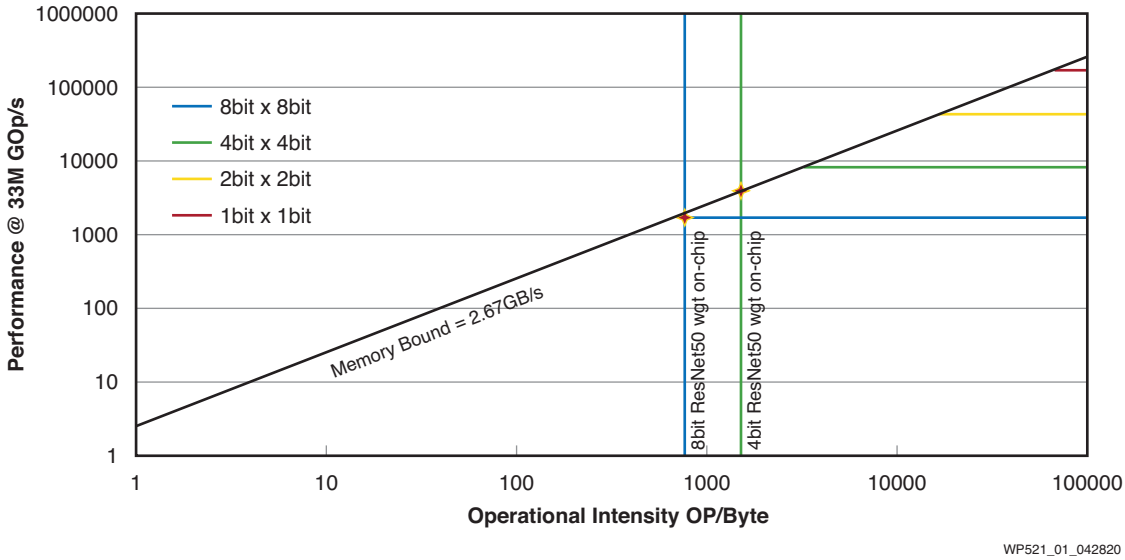

Figure 1: **Rooflines Model for ZCU102 across Different Bit-Widths**

How to Quantize a Full-Process Hardware-Friendly CNN

To make the whole quantization process hardware-friendly, the INT4 quantization method can be split into three categories: quantization mechanism, hardware-friendly quantization design, and quantization-aware training.

Quantization Mechanism

Xilinx used Trained Quantization Thresholds (TQT) [Ref 7] to convert the DNN from single precision floating point (FP32) to INT4. For weights and activations, the quantization function can be formally written as:

$$q(x;s) := clip\left(\left\lfloor\frac{x}{s}\right\rfloor;n,p\right)\cdot s$$

Equation 1

where $n=-2^{b-1}, p=2^{b-1}-1$ and $s=\frac{2^{\lceil \log_2 t\rceil}}{2^{b-1}}$ for signed data; $n=0, p=2^{b}-1$ and $s=\frac{2^{\lceil \log_2 t\rceil}}{2^{b}}$ for unsigned data.

Equation 1 shows that the quantization value of input x depends on threshold t, bit-width b, and quantization scale factor s. The threshold t is usually initialized to the maximum value of the absolute value of the tensor to be quantized. It is then optimized during training in the form of $\log_2 t$. Quantization scale factor s is power of 2, which is hardware friendly. The clip operation removes some outlier data and encourages a tighter distribution of weights and activations, which is better for quantization.

As mentioned above, $\log_2 t$ is a learnable parameter during training, and it is optimized to find a suitable quantization range. Conversely, the gradient of $\log_2 t$ can be created by the chain rule. The gradient of input x can also be calculated:

$$\nabla_{(\log_2 t)} q(x;s) \coloneqq s \ln 2 \cdot \begin{cases} \left\lfloor \frac{x}{s} \right\rceil - \frac{x}{s} & if\ n \le \left\lfloor \frac{x}{s} \right\rceil \le p \\ n & if \left\lfloor \frac{x}{s} \right\rceil < n \\ p & if \left\lfloor \frac{x}{s} \right\rceil > p \end{cases}$$

Equation 2

$$\nabla_x q(x;s) \coloneqq \begin{cases} 1 & if\ n \le \left\lfloor \frac{x}{s} \right\rceil \le p \\ 0 & otherwise \end{cases}$$

Equation 3

For $\lfloor x \rceil$ (round) and $\lceil x \rceil$ (ceil), non-differentiable function, STE is used to get the gradient, defined in Equation 4.

$$\frac{\partial}{\partial x} \lfloor x \rceil = \frac{\partial}{\partial x} \lceil x \rceil = 1$$

Equation 4

TQT proves that the log representation can ensure scale invariance of threshold and input. Training thresholds are more manageable in the log domain and prove very efficient.

Hardware-Friendly Quantization Design

A low-bit network must be built from a full-precision network for quantization training. Based on full-process hardware-friendly quantization, following are some popular network structures and summaries of several coarse-grained modules' quantization solutions. Through these quantization modules, the INT4 quantization method can be used for a variety of network structures. Quantization solutions of some common modules are shown in Figure 2. The dotted line in Figure 2 means that it can be added or deleted according to the actual network structure.

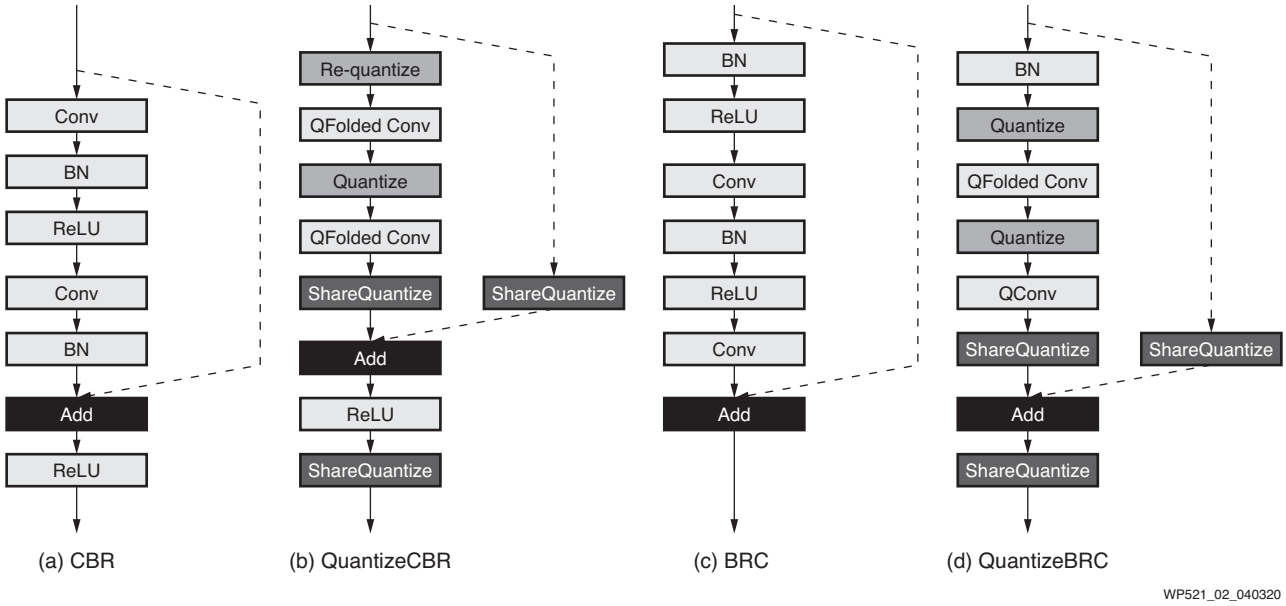

Figure 2: **Module Quantization**

Module 1: CBR(Conv+BN+ReLU)

As a common structure in CNN, the BN layer is merged for reducing flops during training and inference. However, inconsistency exists for the BN layer; the batch operation uses the current batch's mean and variance during training and moves the mean and variance during inference. If the merged parameters obtained from the mean and variance of the current batch are quantized, it leads to deviations when inferenced. To eliminate this mismatch, the following best practices [Ref 8], [Ref 9] should be used to quantize this structure. After folding BN to Conv, the folded parameters to INT4 are quantized. This module's quantization is shown in the Figure 2(b).

Module 2: BRC(BN+ReLU+Conv)

As shown in Figure 2(c), when the BN layer following the convolutional layer is merged, there is still an independent BN layer. However, in the existing INT4 quantization methods, the BN layer has received little attention. To effectively deploy this independent BN layer, a streamlined deployment for quantized neural networks [Ref 10] is used to keep the full-precision during training and absorb float scale and bias to threshold during inference. This method can be extended to all linear operations including convolution in inference while maintaining the accuracy. This module's quantization is shown in the Figure 2(d).

Module 3: Add

For the Add module, the shared quantization layer is used to quantize all inputs and outputs. The Add operation is sensitive to the precision based on experience, but it takes fewer hardware resources. So, this layer is usually quantized to 8-bit. Furthermore, a scale-sharing policy is used to quantize all inputs and outputs. The shared policy enables the hardware to bypass scale calculations, eliminating the floating-point multiplication need. As shown in Figure 2(b) "ShareQuantize" means that these quantization layers share the same scale.

Others:

To ensure that the input of the Conv operation is 4 bits, 8 bits output of add operation needs to be quantized to 4 bits again, as shown by ``Re-quantize" in Figure 2. For the first and last layers, INT4 quantization is still performed. The output of the entire network is quantized into 8 bits. The innerproduct layer is consistent with the convolutional layer.

Quantization-Aware Training

Quantization-aware training is typically used as a key technology to reduce the accuracy gap between low-bit and full-precision. In the INT4 quantization method described in this white paper, it still plays an indispensable role. The quantization-aware training process uses Algorithm 1 (shown below).

Algorithm 1: Layer-Wise Quantization-Aware Training

Input:

full-precision input, weights and bias: *X, W, Bias*

the learnable log-domain threshold for input and weights: a_x, a_w, a_{bias}

bit-width: for input and weights, b=4; for bias, b=8

Output:

the output: Y

1. Initialize $a_x = log_2\, max\ (|x|)$, $a_w = log_2\, max\ (|w|)$, $a_{bias} = log_2\, max\ (|bias|)$

2. Compute $q(x)$, $q(w)$, and $q(bias)$ according to Equation 1

3. $Y = Forward(q(x), q(w), q(bias))$

4. Compute the classification loss: *Loss*. Regularization techniques are used for all learnable parameters.

5.Refer to Equation 3

$$\nabla_X L = \frac{\partial L}{\partial q\,(x)} * \frac{\partial q\,(x)}{\partial X}, \frac{\partial q\,(x)}{\partial X}$$

6. Update full-precision parameters using Adam

INT4 Optimization on Xilinx DSP Slices

A multiplication and accumulation (MAC) operation can be achieved by using the DSP hardware resources. With optimization, the DSP can handle as many MAC operations as possible on 16nm or 28nm devices. Taking 16nm as an example, the UltraScale™ architecture's DSP48E2 slices in Xilinx programmable devices are dedicated slices [Ref 11]. DSP48E2 slice consists of a 27 x 18 twos complement multiplier and a 48-bit accumulator. As shown in Figure 3, MAC can be done with Xilinx DSP slices.

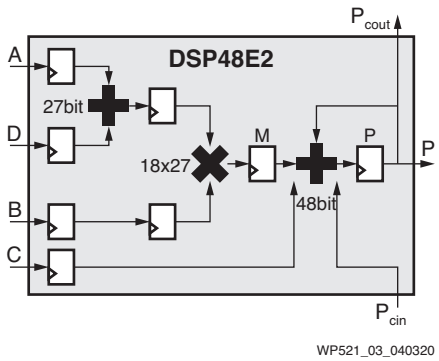

Figure 3: **DSP48E2 Slice in MAC Mode**

INT4 Optimization

In low-precision MAC operations, multiplication is *a*b*, where *a* is 4-bit unsigned activation data and *b* is a 4-bit signed weight data. The DSP48E2 slice can be configured for a 4 channel multiplication operation, as shown in Figure 4.

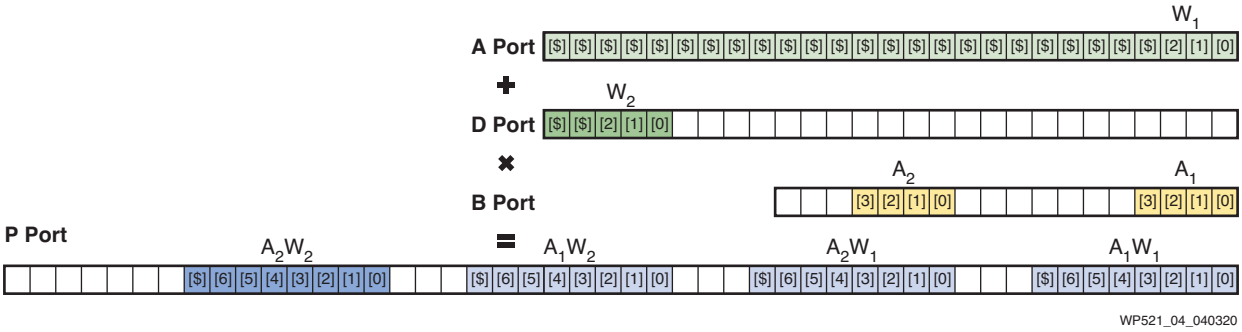

Figure 4: **DSP48E2 Configuration Mode for 4-Channel Packing**

Port A of the DSP48E2 slice is 27-bits wide. Port B is 18-bits wide. A multiplication of *int4 * uint4* generates a result that needs at least an 8-bit width. The premise of making full use of the DSP resources is to ensure that the output remains correct when multiple multiplications are packaged together. To ensure this, guards are added between the channels. When four channels of MAC are packed together, enough guard bits need to be placed between the two inputs. According to the DSP48E2 slice's design, the guard-bit is set to be 3 bits:

$$(A_2 \cdot 2^{11} + A_1) \cdot (W_2 \cdot 2^{22} + W_1) = A_2W_2 \cdot 2^{33} + A_1W_2 \cdot 2^{22} + A_2W_1 \cdot 2^{11} + A_1W_1$$

Equation 5

The first channel A_1*W_1 is placed on the 4 LSBs in their respective ports. The next channel A_2*W_1 need to be shifted at least 8-bits for correct calculations. The second channel shares weight data W_1 with the first. A_2 is shifted 11 bits in Port B. The 3-bit guard is for maximizing the use of DSP resources. The last element W_2 is assigned to Port A. The last two channels are A_1*W_2 and A_2*W_2. The weight is signed data. Before the multiplication, two weight data are packed via a 27-bit pre-adder. W_2 cannot be placed on the four MSBs of the D Port because W_1 requires sign extension [Ref 12]. If W_2 is in the MSBs, the pre-adder will overflow when the W_1 < 0 and W_2 = –8. The post 48-bit adder can be used as an accumulator, adding the previous level DSP results by cascading. In this way, a single DSP48E2 can achieve MAC of four channels in a single clock cycle.

The bit-width of the result becomes higher after the accumulation. The hardware-friendly quantizer is a set of shift registers, and it can control the number of bits shifted by the instruction. Shift operation is hardware friendly. In low-precision CNN, convolution can use one of two kinds of quantization. One is to output 8-bit for element-wise. The other is to output 4-bit for the next convolution. By algorithmic optimization, both quantization methods can be quantized as a step of 2^k. The difference is the bit-width of output data and whether these are signed data.

Enhanced Usage of DSP

The DSP double data rate (DDR) technique is used to improve the performance achieved with the DSP48 slice [Ref 13]. Therefore, two input clocks for the DPU are needed; one for general logic and the other for DSP slices. The difference between a DPU not using the DSP DDR technique and a DPU with enhanced usage architecture is shown in Figure 5.

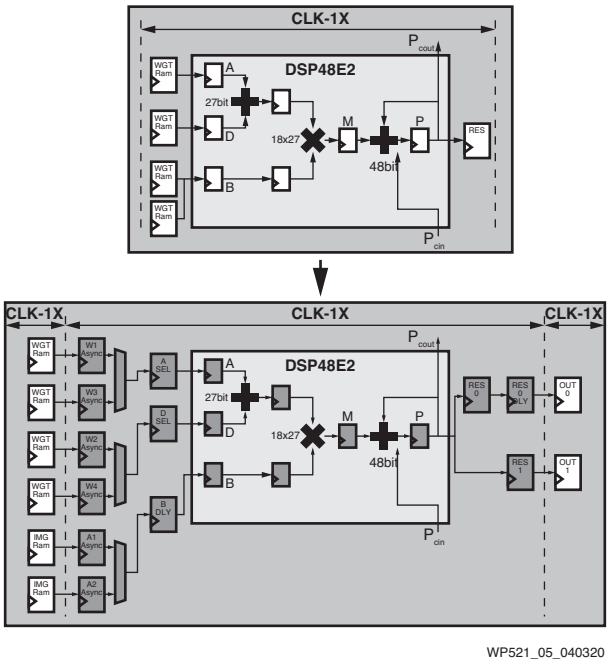

Figure 5: **Difference between DSP without DDR and DSP Enhanced Usage**

Compute Map to CNN Requirements

Convolution is the main computing requirement of a CNN network. The actual calculation tasks for convolution are as follows:

$$X_f = \sum_{n=0}^{N} A_{nf} \cdot W_{nf} + Bias_f$$

Equation 6

where A_{nf} is a floating point feature map, and W_{nf} is a floating point weight. Its essence is a MAC operation. According to Xilinx's novel quantization-aware training solution, the convolution calculation of floating points is quantized as the following:

$$X_f = \sum_{n=0}^{N} \alpha_{xf}\, A_{int} \cdot \alpha_{wf} W_{int} + \alpha_{bf} Bias_{int}$$

Equation 7

$$X_{fixed} = 2^k \left(\sum_{n=0}^{N} A_{int} \cdot W_{int} + 2^{j-k} Bias_{int} \right)$$

Equation 8

where a_{xf}, a_{wf}, and a_{bf} are scale. These floating-point parameters are converted into $2^k * 2^k$, which is a hardware-friendly scale that can be easily implemented by using a shift operation on an FPGA.

The DSP block needs two weights and two features in one clock cycle. Each of them can be shared as shown in Figure 6.

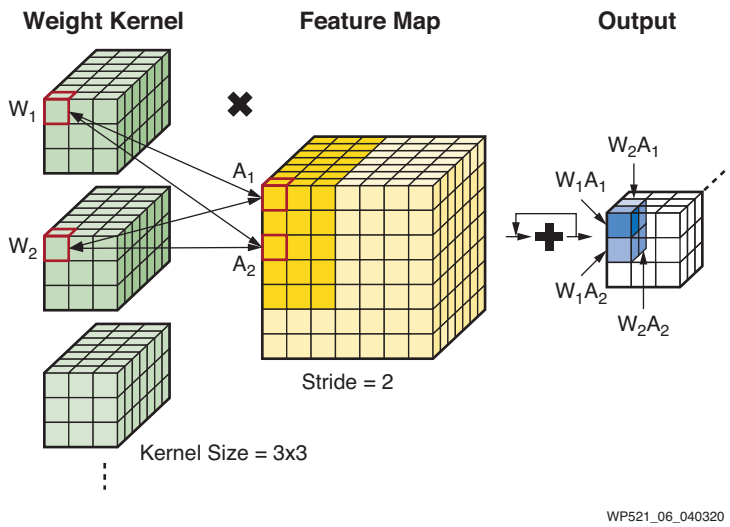

Figure 6: **Convolution Calculation Tasks and How to Share Multipliers**

In the kernel where W_1 is located, all pixels of kernelwidth*kernelhigh*channel need to be multiplied by the feature and added up to get one pixel of the output. In the same layer, each weight kernel shares the same feature map. The two weights that are packed need to come from two different weight kernels. When a weight kernel slides on a feature map, the corresponding feature data needs to be multiplied by the same weight kernel in each step. The two features in the one DSP48 block should come from the different sliding windows in the same feature map.

Model Quantization and Performance Simulation

The following sections describe CV tasks used in quantization-aware training. These include image classification, pose estimation, 2D detection, 3D detection, semantic segmentation, and multi-task.

Benchmark Classification Models

Experiments on the ImageNet classification dataset were conducted with the following results. The networks include ResNet50-V1, ResNet50-V2. For all experiments, the dataset is finetuned from the float model. All bias parameters are quantized to 8-bit. Experiment results are listed in Table 1.

Table 1: **ResNet50-like Network Accuracy with Different Bit-Widths**

Model	A / W	Input / Output	First Layer / Last Layer	Element-Wise	Top 1	Top 5
ResNet50V1	float	float	float	float	76.15	92.87
	8/8	8/8	8/8	8	76.024	92.940
	4/4	4/8	4/4	8	74.588	91.998
ResNet50V2	float	float	float	float	77.596	93.53
	8/8	8/8	8/8	8	77.474	93.536
	4/4	4/8	4/4	8	74.124	91.422

The benchmark classification model results in Table 1 show the effectiveness of this method. Specifically for ResNet50V1, the 4-bit XDPU solution has only a 1.4% gap between the 8-bit XDPU solution on Top 1 accuracy and a 0.9% gap to 8-bit XDPU solution on Top 5 accuracy.

Real-World ADAS Models Including Pose Estimation, Detection, Segmentation, Multi-task, Etc.

To further verify the generality of the quantization method, other cv tasks were conducted in real scenarios.

Pose Estimation

The pose estimation task uses the more complex Stacked Hourglass network [Ref 14]. The accuracy of two network structures with layer-wise mode in the pose estimation experiment on MPII [Ref 15] dataset is evaluated. The results are listed in Table 2.

Table 2: **Hourglass Network Accuracy with Different Bit-Width**

Model	A / W	Input / Output	First Layer / Last Layer	Element-wise	Accuracy (%)
hg-s2-b1	**float**	**float**	**float**	**float**	**71.29**
	8/8	8/8	8/8	8	72.04
	4/4	4/8	4/4	8	69.67
hg-s8-b1	**float**	**float**	**float**	**float**	**83.46**
	8/8	8/8	8/8	8	83.06
	4/4	4/8	4/4	8	82.51

In Table 2, hg-s2-b1 means that the number of stacks is 2, and the number of blocks is 1. Hg-s8-b1 means that the number of stacks is 8, and the number of blocks is 1. Table 2 shows that the Xilinx INT4 quantization solution achieves a comparable accuracy with float model.

2D Detection

In an ADAS system, the BDD100K [Ref 16] dataset is used for 2D detection. In addition, an FPN structure is added to the ResNet18-SSD and is used as the detection network. The experimental results are shown in Table 3.

Table 3: **Detection Accuracy With Different Bit-Width**

Task	A / W	Input / Output	First layer / Last layer	Element-Wise	mAP@0.5 (%)
2D Detection	float	float	float	float	39.0
	8/8	8/8	8/8	8	39.5
	4/4	4/8	4/4	8	37.8

Table 3 shows that after finetuning, the 8-bit quantization model achieves higher mAP than the float model. With progressive finetuning from 8-bit to 4-bit, the final 4-bit quantization model suffers a mAP loss within 2%. The visualization of 2D detection is shown in Figure 7.

Figure 7: **Visualization of 2D Detection**

3D Detection

In the 3D detection task of the ADAS system, a KITTI dataset [Ref 17] was used. PointPillars [Ref 18] is used to conduct 3D prediction task. The experimental results are shown in Table 4.

Table 4: **3D Detection Results with Different Bit-Width**

Task	A/W	Input/Output	First layer/Last layer	Element-wise	Moderate Car AP@0.5(%)	
					BEV	3D
3D Detection	float	float	float	float	90.12	90.03
	8/8	8/8	8/8	8	90.00	89.84
	4/4	4/8	4/4	8	90.07	89.87

As Table 4 demonstrates, with finetuning tricks, the results of 4-bit quantization model has a gap of only 0.16% compared to the float model. The 3D detection results of 8-bit and 4-bit are illustrated in Figure 8.

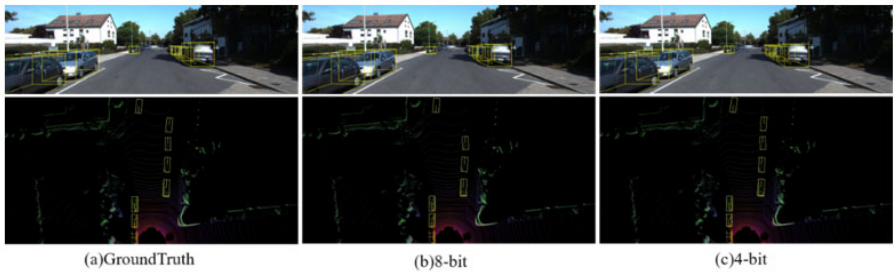

Figure 8: **Visualization of 3D Detection on Camera and Bird-Eye-View**

Semantic Segmentation

In the semantic segmentation task of the ADAS system, the CityScape's dataset [Ref 19] focuses on urban visual scene understanding. The experiments were conducted on a Feature Pyramid Network (FPN) with ResNet18 as the backbone. Results are shown in Table 5.

Table 5: **Semantic Segmentation Accuracy with Different Bit-Width**

Task	A/W	Input/Output	First Layer/ Last Layer	Element-Wise	mIoU(%)
Segmentation	float	float	float	float	63.62
	8/8	8/8	8/8	8	63.97
	4/4	4/8	4/4	8	61.90

Table 5 shows that 8-bit model achieves higher mIoU than float model and 4-bit model only drops 1.7% mIoU. The visualization of semantic segmentation is shown in Figure 9.

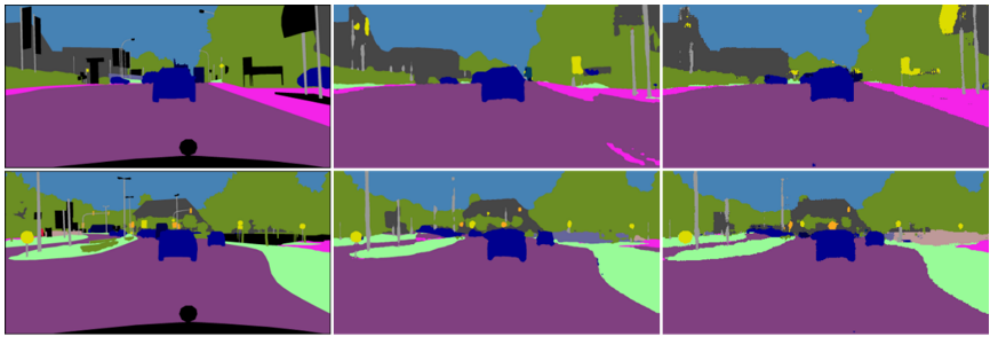

Figure 9: **Visualization of Semantic Segmentation**

Multi-Task Learning

To improve the generalization ability and accuracy of the model, multiple training datasets are used in the multi-task model, including the Waymo and BDD100k for detection, and the BDD100k and Cityscapes [Ref 19] for segmentation. These studies are conducted on a Feature Pyramid Network (FPN) with ResNet18 as the backbone. Results are shown in Table 6.

Table 6: **Multi-Task Accuracy with Different Bit-Widths**

Task	A/W	Input/ Output	First Layer/ Last Layer	Element-Wise	Accuracy (%)	
					Det(mAP)	Seg(mIoU)
Multi-task	float	float	float	float	39.06	42.12
	8/8	8/8	8/8	8	39.94	45.3
	4/4	4/8	4/4	8	37.4	43.91

Table 6 shows that the 8-bit quantization model achieves higher mAP and the same mIoU than the float model. With progressive finetuning, the final 4-bit quantization model reduced mAP by 1.66%, but the mIoU increased by 1.79% compared to float.

Still lower than 8-bit, multi-task visualization results are shown in Table 10.

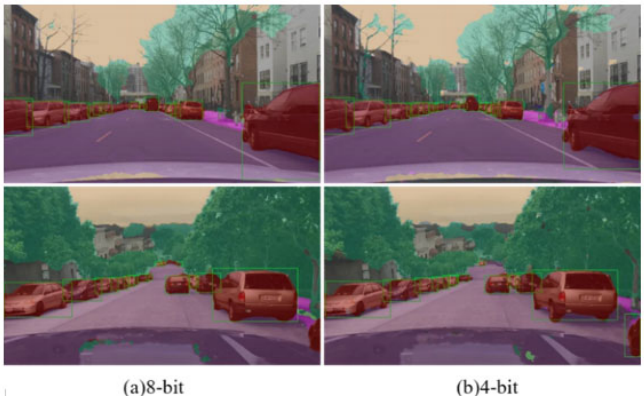

Figure 10: **Visualization of Multi-Task Learning**

Competitive Analysis: 8-Bit vs. 4-Bit

The 4-bit XDPU runs at 300MHz frequency on the following three evaluation boards: Ultra96 and Zynq UltraScale+ MPSoC ZCU104 and ZCU102. Table 7 shows the performance comparison between 4-bit XDPU and 8-bit XDPU. On different FPGAs, 4-bit XDPU achieved performance improvements ranging from 1.5X to 2.0X. For example, the hardware resources used by the ZCU102 board did not increase, but the performance increased by 2X.

Table 7: **Performance Comparison between 4-Bit XDPU and 8-Bit XDPU**

	Ultra96	ZCU104	ZCU102
8-Bit XDPU	691GOPs	2.45TOPs	3.69TOPs
4-Bit XDPU	1228GOPs	3.69TOPs	7.37TOPs

For the two accelerators with different precision, resources were compared after all functions, such as pooling, element wise, depth-wise convolution and average pooling, are enabled. As shown in the Table 8, the usage of DSP and RAM decreased significantly under the same performance architecture. Due to the reduced resource consumption, the 4-bit XDPU architecture was expanded to the maximum size of B8192. Using the B8192 architecture allows higher performance on a single device.

Table 8: **Resource Comparison between 4-Bit XDPU and 8-Bit XDPU**

4-Bit XDPU					8-Bit XDPU				
Arch	LUTs	Regs	Block RAM	DSP	Arch	LUTs	Regs	Block RAM	DSP
B512 (4x 8x 8)	25322	32211	41.5	62	B512 (4x 8x 8)	26482	33530	73.5	110
B800 (4x10x10)	29137	38398	56	97	B800 (4x10x10)	29711	40184	91.5	157
B1024 (8x 8x 8)	31378	42699	57.5	122	B1024 (8x 8x 8)	32598	47282	105.5	218
B1152 (4x12x12)	32928	43337	73	116	B1152 (4x12x12)	31769	46462	123	212
B1600 (8x10x10)	36504	52101	76	192	B1600 (8x10x10)	36838	58204	127.5	312

Table 8: **Resource Comparison between 4-Bit XDPU and 8-Bit XDPU *(Cont'd)***

4-Bit XDPU					8-Bit XDPU				
B2304 (8x12x12)	38389	58090	97	230	B2304 (8x12x12)	40039	68469	167	422
B3136 (8x14x14)	43316	67901	120	324	B3136 (8x14x14)	43972	79141	210	548
B4096 (8x16x16)	48232	75896	145.5	370	B4096 (8x16x16)	49754	97882	257	690
B8192 (8x32x16)	55890	91426	201.5	690	B8192 (8x32x16)	Unsupported			

Taking the 2D detection model of 13.6 FLOPs in Table 3 as an example, the two precision models 4/4 and 8/8 were tested using 4-bit XDPU and 8-bit XDPU respectively. The computing requirements for this network are 13.6GOP. The frame rate for a 2D detection network is shown in Table 9, and the test does not include pre- and post-processing. Performance and frame rates are not linear due to differences in efficiency and network types. As shown in Table 9, the frame rate of 4-bit XDPU is better than 8-bit XDPU on all platforms.

Table 9: **Frame Rate between 4-Bit DPU and 8-Bit DPU**

	Ultra96	ZCU104	ZCU102
2D Detection (8/8)	30fps	101fps	151fps
2D Detection (4/4)	53fps	145fps	230fps

Conclusion

This white paper explores a low-precision accelerator for CNN with a full-process, hardware-friendly quantization solution on the Zynq UltraScale+ MPSoC and Zynq-7000 SoC family (16nm and 28nm). It describes how INT4 optimized on a Xilinx DSP slice can finish a 4-channel INT4 multiplication in one clock cycle. The packing of DSP meets the computation requirements of convolution. INT4 optimization with DSP can deliver up to 2X peak GOPS and up to 1.77X performance on real hardware over the INT8 XDPU solution. The Xilinx solution achieves comparable results to floating-point models on a variety of CV tasks. For resource-limited and power-constrained use cases, Xilinx is continuing to innovate on new hardware and soft-ware co-optimization methodologies to accelerate deep learning applications.

Acknowledgment

The following Xilinx employees have authored or contributed to this white paper: Tiantian Han (Software Engineer, AI Algorithm), Tianyu Zhang (Design Engineer 2, AI Edge-Computing), Dong Li (Senior Software Development Manager, AI Algorithm), Guangdong Liu (Design Engineer, AI Edge-Computing), Lu Tian (Software Development Director, AI Algorithm), Dongliang Xie (Design Engineer Director, AI Edge-Computing), and Yi Shan (Senior Director, AI).

References

1. Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, et al. 2015. "Imagenet Large Scale Visual Recognition Challenge." International Journal of Computer Vision 115 (3): 211-252.
2. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016a. "Deep Residual Learning for Image Recognition." In Proceedings of the Ieee Conference on Computer Vision and Pattern Recognition, 770-778.
3. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016b. "Identity Mappings in Deep Residual Networks." In European Conference on Computer Vision, 630-645. Springer.
4. Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. "Mobilenets: Efficient Convolutional Neural Networks for Mobile Vision Applications." arXiv Preprint arXiv:1704.04861.
5. Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. "Mobilenetv2: Inverted Residuals and Linear Bottlenecks." In Proceedings of the Ieee Conference on Computer Vision and Pattern Recognition, 4510-4520.
6. Williams, S., 2009. Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore.
7. Sambhav R Jain, Albert Gural, Michael Wu, and Chris Dick. 2019. "Trained Quantization Thresholds For Accurate And Efficient Fixed-Point Inference Of Deep Neural Networks." arXiv Preprint arXiv:1903.08066.
8. Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2704-2713.
9. Raghuraman Krishnamoorthi. 2018. "Quantizing Deep Convolutional Networks for Efficient Inference: A Whitepaper." http://arxiv.org/abs/1806.08342.
10. Yaman Umuroglu, and Magnus Jahre. 2017. "Streamlined Deployment for Quantized Neural Networks." arXiv Preprint arXiv:1709.04060.
11. Xilinx Inc. UltraScale Architecture DSP Slice User Guide, May 2019. https://www.xilinx.com/support/documentation/user_guides/ug579-ultrascale-dsp.pdf.
12. Yao Fu, Ephrem Wu, Ashish Sirasao, Sedny Attia, Kamran Khan, and Ralph Wittig. 2016. "Deep Learning with Int8 Optimization on Xilinx Devices." White Paper.
13. Xilinx Inc. DPU for Convolutional Neural Network v3.0 IP Product Guide, Aug 2019. https://www.xilinx.com/support/documentation/ip_documentation/dpu/v3_0/pg338-dpu.pdf.
14. Alejandro Newell, Kaiyu Yang, and Jia Deng. 2016. "Stacked Hourglass Networks for Human Pose Estimation." In European Conference on Computer Vision, 483-499. Springer.
15. Mykhaylo Andriluka, Leonid Pishchulin, Peter Gehler, and Bernt Schiele. 2014. "2d Human Pose Estimation: New Benchmark and State of the Art Analysis." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 3686-3693.
16. Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. 2018. "Bdd100k: A Diverse Driving Video Database with Scalable Annotation Tooling." arXiv Preprint arXiv:1805.04687.
17. Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. 2013. "Vision Meets Robotics: The Kitti Dataset." The International Journal of Robotics Research 32 (11): 1231-1237.

18. Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. 2019. "PointPillars: Fast Encoders for Object Detection from Point Clouds." In the IEEE Conference on Computer Vision and Pattern Recognition.
19. Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. 2016. "The Cityscapes Dataset for Semantic Urban Scene Understanding." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 3213-3223.

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
06/24/2020	1.0.1	Typographical edit.
06/19/2020	1.0	Initial Xilinx release.

Disclaimer

Automotive Applications Disclaimer