electronics

MDPI

Article

HBCA: A Toolchain for High-Accuracy Branch-Fused CNN Accelerator on FPGA with Dual-Decimal-Fused Technique

Zhengjie Li [1], Lingli Hou [2], Xinxuan Tao [1], Jian Wang [1] and Jinmei Lai [1,*]

[1] School of Microelectronics, Fudan University, No. 825, Zhangheng Road, Pudong New Area, Shanghai 201203, China
[2] Chengdu Sino Microelectronic Technology Co., Ltd., Chengdu 610041, China
* Correspondence: jmlai@fudan.edu.cn

Abstract: The programmability of FPGA suits the constantly changing convolutional neural network (CNN). However, several challenges arise when the previous FPGA-based accelerators update CNN. Firstly, although the model of RepVGG can balance accuracy and speed, it solely supports two types of kernels. Meanwhile, 8-bit integer-only quantization of PyTorch which can support various CNNs is seldom successfully supported by the FPGA-based accelerators. In addition, Winograd F(4×4, 3×3) uses less multiplication, but its transformation matrix contains irregular decimals, which could lead to accuracy problems. To tackle these issues, this paper proposes High-accuracy Branch-fused CNN Accelerator (HBCA): a toolchain and corresponding FPGA-based accelerator. The toolchain proposes inception-based branch–fused technique, which can support more types of kernels. Meanwhile, the accelerator proposes Winograd-quantization dual decimal–fuse techniques to balance accuracy and speed. In addition, this accelerator supports multi-types of kernels and proposes Winograd decomposed-part reuse, multi-mode BRAM & DSP and data reuse to increase power efficiency. Experiments show that HBCA is capable of supporting seven CNNs with different types of kernels and more branches. The accuracy loss is within 0.1% when compared to the quantized model. Furthermore, the power efficiency (GOPS/W) of Inception, ResNet and VGG is up to 226.6, 188.1 and 197.7, which are better than other FPGA-based CNN accelerators.

Keywords: CNN; FPGA; branch-fused; Winograd-quantization-dual-decimal-fuse

Citation: Li, Z.; Hou, L.; Tao, X.; Wang, J.; Lai, J. HBCA: A Toolchain for High-Accuracy Branch-Fused CNN Accelerator on FPGA with Dual-Decimal-Fused Technique. *Electronics* **2023**, *12*, 192. https://doi.org/10.3390/electronics12010192

Academic Editors: D. J. Lee and Dong Zhang

Received: 31 October 2022
Revised: 13 December 2022
Accepted: 26 December 2022
Published: 30 December 2022

1. Introduction

Convolutional neural networks (CNNs) excel in computer vision. ResNet [1] exceeds human-level accuracy with a top-five error rate below 5%. In order to gain higher accuracy performance, the architectures of CNNs are constantly evolving [2], such as AlexNet [3], RestNet [1], SENet [4], and RepVGG [5]. The architecture of CNN can be obtained by means of autonomous search [6] or manual design [7]. Complicated CNN architecture can obtain higher accuracy, but compared to simple CNN architecture, it decreases speed. RepVGG [5] has multi-branches architecture at the training stage which achieves greater accuracy, and removes side branches at the inference stage, which increases speed.

In recent years, more applications have needed CNN acceleration, such as collision-avoiding drones, autonomous vehicles, medical image diagnostics, and failure detection in production lines [8,9]. For CNN acceleration, accuracy is important, especially in autonomous vehicles. There are many training techniques [10] designed to increase training accuracy, such as simulated situation [11] and federated learning [12]. When deriving one CNN with greater accuracy, the original CNN deployed in the accelerator with an unsatisfactory accuracy performance should be instantly replaced by this improved CNN for building an excellent accelerator to potentially satisfy the strict requirements of real-world applications.

Compared with CPU, GPU and ASIC, FPGA has the overall advantage of high speed, low power, and programmability [13]. In particular, the programmability of FPGA is suitable for CNN updating. Therefore, FPGA has become an appealing platform to accelerate CNNs [14]. Unfortunately, designing with FPGAs is a complex task requiring hardware expertise and it is difficult for CNN software engineers to use; therefore, a CNN-to-FPGA toolchain is proposed to tackle this issue. However, in the existing FPGA-based accelerator research, most toolchains need to regenerate HDL when updating CNN, which means re-synthesis and re-implementation [15]. For different timing constraints, it takes several hours or days to generate a bitstream. It always takes several rounds of iteration of place and route to meet all timing constraints, thereby resulting in an extremely long search and development time. In addition, if the timing constraints are not met, the frequency must be decreased, which inevitably leads to performance deterioration. Especially if the utilized logic resources are significantly increased, this may lead to routing failure. Accordingly, when one accelerator updates CNN, reliability and a short development time are vital for achieving outstanding performance. Numerous efforts have been devoted to achieving this goal. Firstly, OPU [16] and Light-OPU [17] both read parameters from an off-chip memory to update CNN, which results in additional consumed power, latency, and hardware costs. This, as for resource-limited edge devices, compression techniques [18] and TinyML models [19,20] makes it possible to store all CNN parameters in an on-chip memory. UpdateMEM utility [21] generates bitstream without a re-synthesis and re-implementation process, which decreases the development time and eliminates timing and routing issues. The architecture of CNN may change when we are updating it, such as the kernel size, which means the accelerator needs to support more types of kernels. Although the RepVGG can strike a balance between accuracy and speed, it still suffers from merely supporting two types of kernels.

Hardware-oriented optimization methods reduce the complexity of the computation and consequently increase power efficiency [9]. In addition, CNN acceleration techniques that improve power efficiency and throughput without sacrificing accuracy or inducing additional hardware costs are critical [22].

Original CNNs are typically 32-bit float models. During inference, a quantization technique is widely applied to decrease CNN parameters and computation resources. An 8-bit fix-point quantization scheme has been widely employed in FPGA-based CNN accelerators, but there are two major challenges. Firstly, 8-bit static fix-point quantization may lead to large accuracy degradations. The 16-bit static fix-point quantization used in [23] makes VGG accuracy decrease from 88.00% to 87.94%. But the 8-bit static fix-point quantization of the VGG of paper [23] fails because activation values of fully connected layer are zeros. In addition, 8-bit dynamic fix-points may have different effects to different CNNs. The 8-bit dynamic fix-point quantization of paper [24] makes GoogLeNet accuracy decrease by 7.63%, from 93.33% to 85.70%; while SqueezeNet accuracy increases by 0.02%, from 80.3% to 80.32%. Recently, PyTorch provided 8-bit integer only quantization, which improved the tradeoff between accuracy and speed. It adopts per-channel and per-layer quantization, and supports various CNN architectures, and the accuracy loss is within 1% [25,26]. But the quantized model of PyTorch does not directly support FPGA development. Paper [27] uses similar quantization methods, but it does not create a toolchain, and does not implement all functions of the low-precision general matrix multiplication library, which decreases accuracy.

CNN is computation-intensive [28]. Fast algorithms can reduce arithmetic complexity and enhance power efficiency [29]. Many FPGA-based CNN accelerators utilize Winograd fast algorithms [30–32]. Winograd F($2 \times 2, 3 \times 3$) is widely used, because its transformation matrix is simple. Furthermore, Winograd F($4 \times 4, 3 \times 3$) can further reduce the number of multiplications, but its transformation matrix contains irregular decimals, such as $1/24$, $1/12$, and $1/6$, which cannot be transformed into shift operations. Paper [33] finds that the width of the fraction has distinct impacts on accuracy when using Winograd F($4 \times 4, 3 \times 3$). When the width of the fraction is 13, 12, 11, and 10, the accuracy of the VGG decreases by

4.02%, 21.41%, 97.43% and 100%, respectively. The original Winograd only supports one type of kernel whose size and stride are 3×3 and 1. Hence, paper [31] solely supports this type of kernel. Paper [34] proposes the decomposable Winograd method (DWM), which expands the usage range of Winograd to other types of kernels. Paper [30] proposes a similar method for the stride of 2 kernel, and validates two types of kernels on FPGA. In addition, paper [30] considers how to save Look-Up Table (LUT) resources when supporting two types of kernels. Paper [35] proposes a more efficient approach for the kernel size of 3×3 and stride of 2 and eventually saves 49.7% of LUT resources compared with paper [30]. But papers [30,35] neglect considering how to save LUT resources when types of kernels are larger than 2.

If the FPGA-based CNN accelerator adopts Winograd, the DSP resources are not critical [31]. Paper [30,31,33] uses the $(A \times B)$ function of DSP to complete element-wise matrix multiplication (EWMM). Paper [36] uses the $((A + D) \times B)$ function of DSP to complete EWMM and partial transformation computation. DSP additionally has cascade, multiply-accumulate, pre-adder, and dynamic reconfiguration functions. Not fully exploring these functions will lead to consuming more LUT resources. In order to increase the memory bandwidth of the reading activation value, paper [31] uses registers to store intermediate activation values. It uses abundant general programmable logic compared with BRAM, which leads to high power.

A data reuse technique is used to decrease the access number of the off-chip/on-chip memory in order to optimize power consumptions. There are two kinds of data reuse techniques: temporal reuse and spatial reuse [37]. Spatial reuse is widely utilized to generate a multiple output feature map (OFM). Paper [31] uses the overlap and save technique, and adopts a temporal reuse of an input feature map (IFM) at the column dimension. But it does not reuse the IFM at the row dimension. Paper [30] also adopts temporal reuse of the IFM when the kernel size is 3×3 and the stride is 1. Padding is important for convolution: paper [31] only supports padding when the kernel size is 3×3 and the stride is 1. Paper [30] does not support padding, which results in accuracy loss. How to implement data spatial reuse, data temporal reuse and padding are not addressed when the accelerator adopts Winograd and supports multi types of kernels.

To deal with the above problems of implementing and updating CNN on an accelerator, we propose High-Accuracy Branch-Fused CNN Accelerator (HBCA): a toolchain and corresponding accelerator. For on-chip CNN model updating, the toolchain generates bitstream without re-synthesis and re-implementation. The toolchain proposes inception-based branch-fuse techniques to support more branches and more types of kernels, which balances accuracy and speed. The accelerator supports PyTorch's 8-bit integer-only quantization and proposes a dual-decimal-fuse technique to balance accuracy and speed. The decimal of the Winograd transformation matrix and the decimal of the scale parameter of the 8-bit integer-only quantization fuse into one; the fused decimal is then transformed into a multiply-and-shift operation, all computation is integer-based, and there is no decimal computation. The accelerator supports data spatial reuse, data temporal reuse and padding when Winograd of multi-types of kernels is adopted. It also proposes the Winograd decomposed-part reuse (WDPR) technique which saves LUT resources, and thus decreases power consumption. The accelerator fully explores functions of BRAM and DSP module of FPGA, which decreases the utilization of general programmable logic resources, and increase power efficiency.

We implement seven CNNs with four types of kernels on a Xilinx XC7V690T FPGA. The accuracy losses of seven CNNs are within 0.1% compared to the quantized models. The power efficiency (GOPS/W) of Inception, ResNet and VGG are 226.6, 188.1 and 197.7, which are better than other FPGA-based CNN accelerators.

The rest of the paper is organized as follows: Section 2 describes the toolchain. Section 3 describes the accelerator. Section 4 presents the experimental results. Section 5 concludes the paper.

2. Toolchain

The architecture of our toolchain is shown in Figure 1. Its inputs are several CNN architectures. The outputs of the toolchain are several corresponding bitstreams. We use PyTorch to train the CNN models.

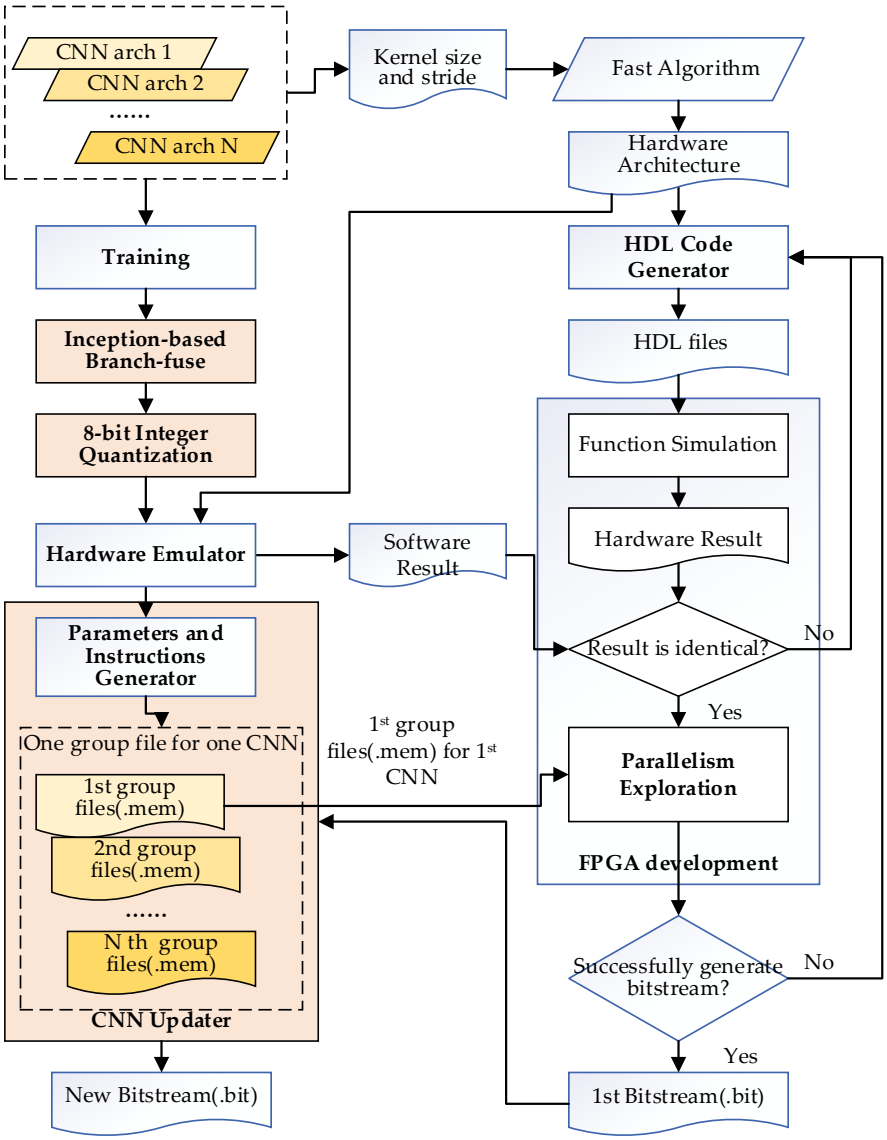

Figure 1. Architecture of toolchain.

- Inception-based branch-fuse

Inception-based branch-fuse technique supports more branches and more types of kernels than RepVGG [5]. The details are discussed in Section 2.1.

- 8-bit integer quantization

We use PyTorch to quantize the inception-based branch-fuse model and obtain an 8-bit integer quantized model. The details are discussed in Section 2.2.

- Hardware emulator

The hardware emulator extracts quantized parameters from the quantized model, and emulates the hardware computation of the convolutional layer, pooling the layer and fully connected layer, and generates software-simulation files. It also achieves accuracy from the test dataset.

- Fast algorithm

From N CNN architectures we can obtain the kernel size and stride. If the kernel size is greater than three, or the kernel stride is larger than one, the decomposable Winograd method [34] of a fast algorithm is adopted. The hardware architecture supports the fast algorithm. For example, hardware architecture includes modules of IFM and CONV weight transformation, as well as EWMM and OFM inversion, which are necessary for Winograd computation.

- HDL code generator

According to hardware architecture in Figure 2, the HDL code generator generates HDL files for all modules in Figure 2 except for the DSP IPs, which are generated by Vivado LogiCore. In particular, all BRAMs are coded using the XPM memory template [21].

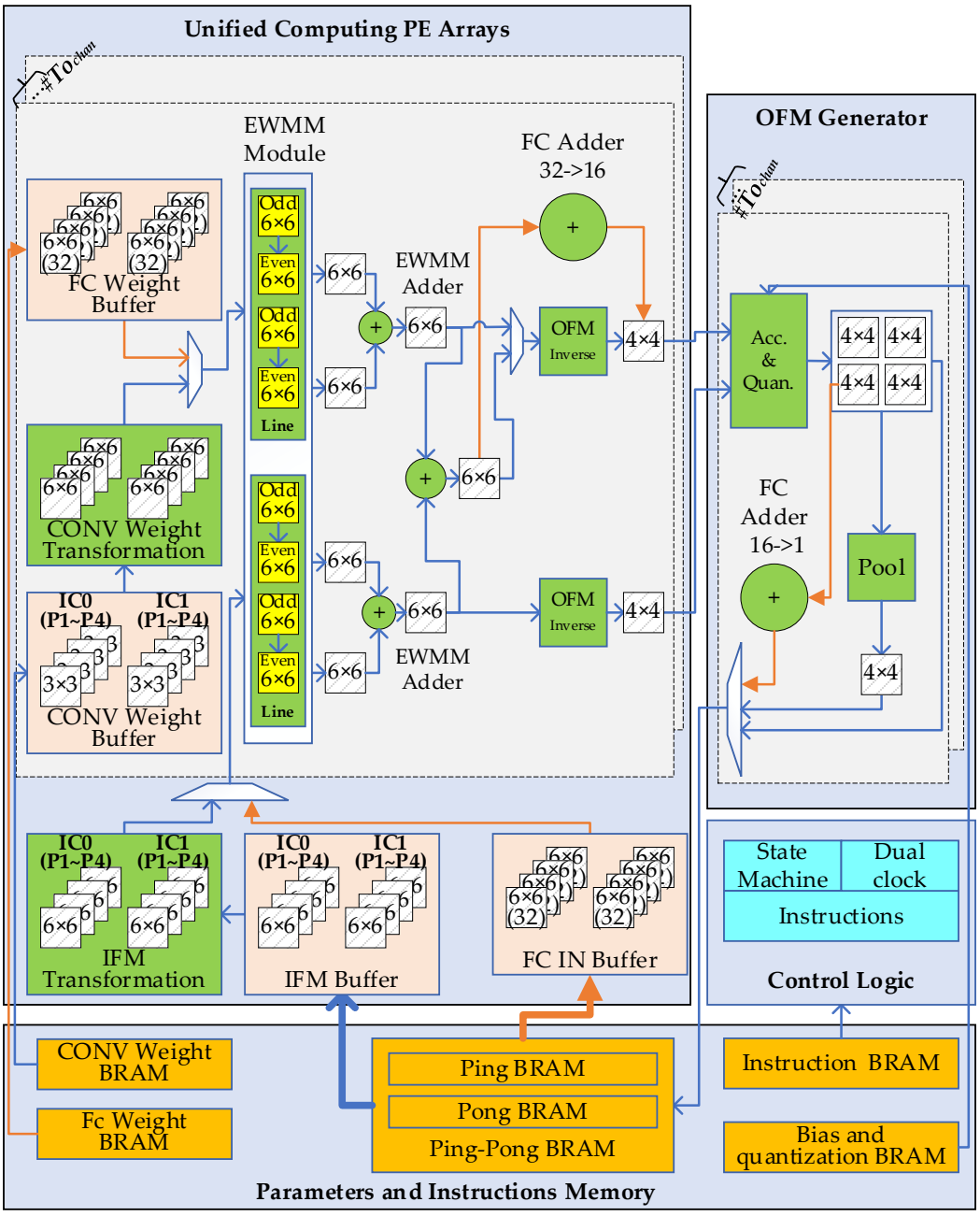

Figure 2. The architecture of the accelerator.

- Function simulation

We simulate HDL files and obtain hardware simulation results. The activation value of each layer is stored in the Ping-Pong BRAM depicted in Figure 2. The content of the Ping-Pong BRAM is written into a hardware-simulation file. The activation value of each layer of hardware emulator is also written into a software-simulation file. We then compare the hardware-simulation file with the software-simulation file. If the two files are different, we modify the HDL code generator. Otherwise, we perform exploration.

- Parallelism exploration

Parallelism exploration identifies the value of To_{Chan} in Figure 2 which represents the number of output channels of the OFM which can be computed at the same time. The larger value of To_{Chan} means more logic resources utility of FPGA. The maximum value of To_{Chan} is bounded by the number of DSPs of FPGA. We gradually decrease To_{Chan} from the maximum value, and the "HDL Code Generator" generates new HDL files according to To_{Chan}. If Vivado can successfully generate bitstream for a certain To_{Chan} value, we can identify the parallelism of the output channels of the OFM. Therefore, we obtain the first bitstream.

- CNN updater

The parameters and instructions generator of the CNN updater generates the initialization file (.mem) of BRAM in the FPGA-based accelerator. The differences of CNNs in our accelerator are parameters and instructions which are stored in BRAMs. If we want to update CNN on FPGA, we use the first bitstream and a group initialization files (.mem) of new CNNs to generate a new bitstream by the *update_mem* command [21] which is listed below.

update_mem -meminfo accelerator.mmi -data ins.mem -proc ins_buffer/xpm_memory_sprom_inst/xpm_memory_base_inst -bit first.bit -out new.bit.

The above commands update the content of the Instruction BRAM *(-proc ins_buffer*) with new instructions (*-data ins.mem*). Together with more -data and -proc pairs and the first bitstream file (*first.bit*), it updates the content of all BRAMs and generates a new bitstream file (*new.bit*). The *update_mem* command avoids the re-synthesis and re-implementation flow, which reduces the development time. By downloading the new bitstream file, the new CNN is updated on the FPGA accelerator.

2.1. Inception-Base Branch-Fuse

We propose the inception-base branch-fuse CNN based on RepVGG and the original inception module. It has two versions: the first one is that the largest kernel size is 3×3 (see Figure 3a,b); the second one is that the largest kernel is 5×5 (see Figure 3c,d).

The means of fusing the branch of the convolution into the backbone of the convolution is similar to RepVGG [5]. However, RepVGG is not capable of supporting the pooling layer. In contrast, we can transfer the average pooling into the equivalent convolution (see Figure 4).

For the 3×3 average pooling described in Figure 4a, $I_{1\text{-}1}$~$I_{3\text{-}3}$ are IFMs, and $O_{1\text{-}1}$ is OFM. The computation details are shown in Equation (1).

$$OFM = (\sum IFM)/9 = (I_{1-1} + I_{1-2} + I_{1-3} + I_{2-1} + I_{2-2} + I_{2-3} + I_{3-1} + I_{3-2} + I_{3-3})/9 \quad (1)$$

The 3×3 average pooling can be transformed into equivalent convolution in Figure 4b. The computation details are shown in Equation (2).

$$OFM = IFM * Kernel = \begin{array}{r} I_{1-1} \times 1/9 + I_{1-2} \times 1/9 + I_{1-3} \times 1/9 + \\ I_{2-1} \times 1/9 + I_{2-2} \times 1/9 + I_{2-3} \times 1/9 + \\ I_{3-1} \times 1/9 + I_{3-2} \times 1/9 + I_{3-3} \times 1/9 \end{array} \quad (2)$$

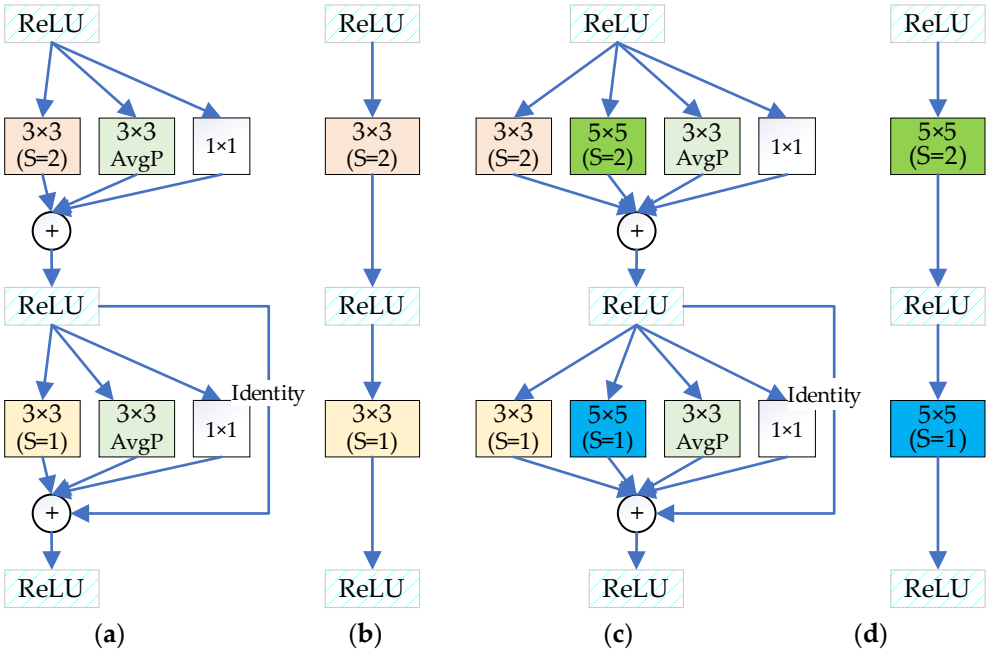

Figure 3. Inception-based branch-fuse. (**a**) Training (3 × 3); (**b**) Inference (3 × 3); (**c**) Training (5 × 5); (**d**) Inference (5 × 5).

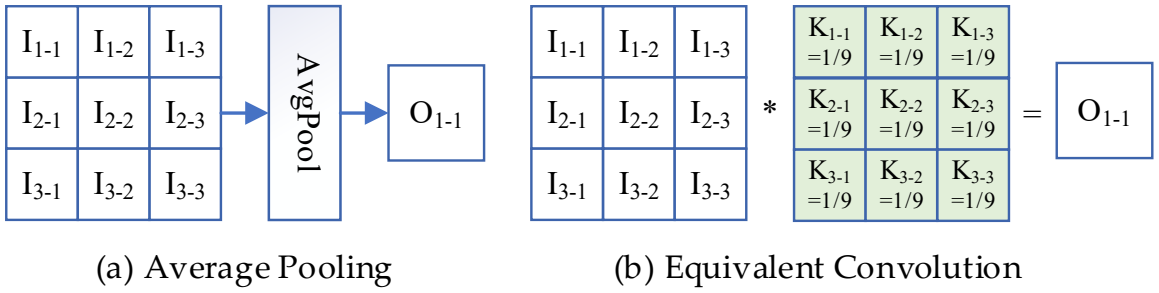

Figure 4. Average pooling is transformed into equivalent convolution.

2.2. The 8-Bit Integer Quantization

PyTorch adopts 8-bit integer-only quantization, which uses the per-channel and per-layer quantization strategy, and supports various CNN architectures with an accuracy loss of below 1% [27,28]. We adopt the same quantization method, and extract quantized parameters and deploy the quantized model on FPGA. First, we calculate *MULT* and *SHIFT* from thescale parameter. In Equation (3), R, S, Z, and Q respectively denote the real number, scale parameter, zero-point, and quantized integer. In Equation (4), R_3 is the real number of an OFM, $R_1[i]$ and $R_2[i]$ are individually real numbers of IFM and the kernel. Based on Equations (3) and (4), we derive Equation (5). S_3, Z_3, Q_3 are the scale, aero-point and quantized integer of OFM. S_1, Z_1, Q_1 are the scale, zero-point and quantized integer of IFM. S_2, Z_2, Q_2 are the scale, zero-point and quantized integer of the kernel.

In Equation (6), S_1, S_2 and S_3 are real numbers, M is the only float-point and in the interval $(0, 1)$. By doubling M until the product is in the interval $[0.5, 1)$, the product is then converted to an approximated fixed-point equivalent value $MULT$. The time of doubling is $SHIFT$. Therefore, M can be represented by a truncated integer multiplier $MULT$ with a right $SHIFT$. $MULT$ and $SHIFT$ are both integers. Equation (5) can be transformed into Equation (7). The computation details are shown in paper [25,26].

$$R = S(Q - Z) \tag{3}$$

$$R_3 = \sum R_1[i]R_2[i] \tag{4}$$

$$Q_3 = Z_3 + \frac{S_1 S_2}{S_3}\left[\sum (Q_1[i] - Z_1)(Q_2[i] - Z_2)\right] \tag{5}$$

$$M = \frac{S_1 S_2}{S_3} \approx 2^{-SHIFT} MULT \tag{6}$$

$$Q_3 = Z_3 + MULT\left[\sum (Q_1[i] - Z_1)(Q_2[i] - Z_2)\right] \gg SHIFT \tag{7}$$

3. Accelerator

A fast algorithm reduces the arithmetic complexity of the convolution, which increases speed. Winograd is one of the fast algorithms, which is more suitable for small kernels. Additionally, the most popular kernel size is 3×3 and 5×5. Therefore, we adopt Winograd to compute the convolution. Meanwhile, we are able to keep the accuracy loss negligible and save LUT resources when supporting multi-types of kernels.

3.1. Dual-Decimal-Fuse Technique

The Winograd convolution includes four modules (see Figure 5a): the IFM transformation, kernel transformation, element-wise matrix multiplication (EWMM) and OFM inversion. Equation (8) is Winograd F (2×2, 3×3), which is widely used, as its transformation matrices are simple (see Equation (9)).

$$OFM_{2\times 2} = A^T_{2\times 4}\left[\left(B^T_{4\times 4} IFM_{4\times 4} B_{4\times 4}\right) \odot \left(G_{4\times 3} Kernel_{3\times 3} G^T_{3\times 4}\right)\right] A_{4\times 2} \tag{8}$$

$$B^T_{4\times 4} = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}, G_{4\times 3} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix}, A_{4\times 2} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & -1 \\ 0 & -1 \end{bmatrix}_{4\times 2} \tag{9}$$

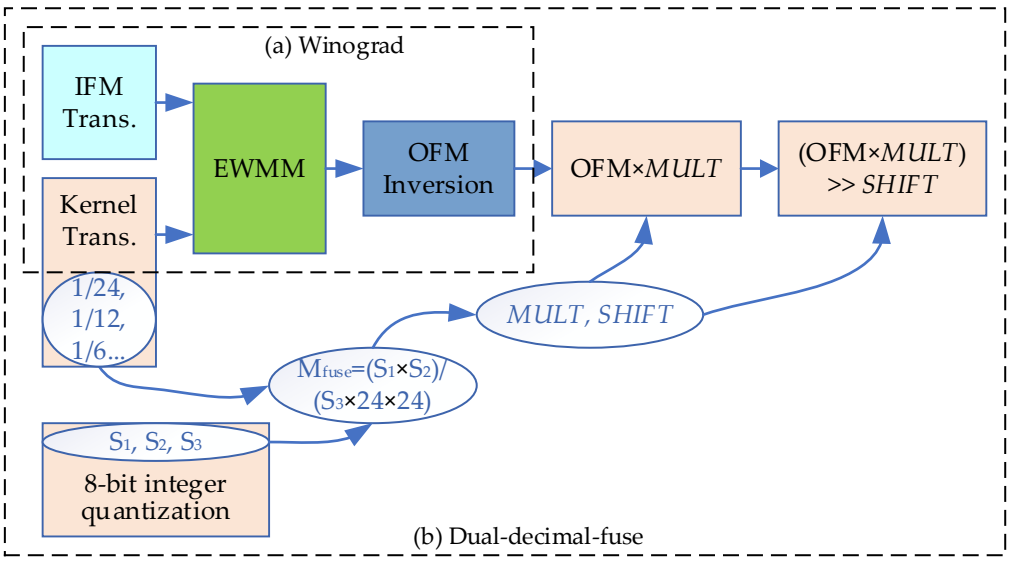

Figure 5. Fast Algorithm (**a**) Winograd (**b**) Dual-decimal-fuse.

In Equation (9), there is $1/2$ in the kernel transformation matrix ($G_{4\times 3}$). Although it is not an integer, it can be converted to the right shift operation.

Equation (10) is Winograd F (4×4, 3×3), which is seldom used because its transformation matrices are complex (see Equation (11)).

$$OFM_{4\times 4} = A^T_{4\times 6}\left[\left(B^T_{6\times 6} IFM_{6\times 6} B_{6\times 6}\right) \odot \left(G_{6\times 3} Kernel_{3\times 3} G^T_{3\times 6}\right)\right] A_{6\times 4} \tag{10}$$

$$B^T_{6\times6}=\begin{bmatrix}4&0&-5&0&1&0\\0&-4&-4&1&1&0\\0&4&-4&-1&1&0\\0&-2&-1&2&1&0\\0&2&-1&-2&1&0\\0&4&0&-5&0&1\end{bmatrix},G_{6\times3}=\begin{bmatrix}\frac{1}{4}&0&0\\-\frac{1}{6}&-\frac{1}{6}&-\frac{1}{6}\\-\frac{1}{6}&\frac{1}{6}&-\frac{1}{6}\\\frac{1}{24}&\frac{1}{12}&\frac{1}{6}\\\frac{1}{24}&-\frac{1}{12}&\frac{1}{6}\\0&0&1\end{bmatrix},A_{6\times4}=\begin{bmatrix}1&0&0&0\\1&1&1&1\\1&-1&1&-1\\1&2&4&8\\1&-2&4&-8\\0&0&0&1\end{bmatrix}_{6\times4} \tag{11}$$

Based on Equation (11), the largest absolute value is equal to 8, and the smallest one is $1/24$. In particular, its kernel transformation matrix contains $1/6$, $1/12$, and $1/24$, which cannot be converted into shift operations. If we do not properly deal with these decimals, the accuracy will decrease.

We propose the dual-decimal-fuse technique to deal with these decimals. The decimals $(1/6, 1/12, 1/24)$ of the kernel transform matrix are fused into decimals of scale (S1, S2, S3) of the 8-bit integer quantization (see Figure 5b). The fused decimal M_{fuse} is transformed into multiply and shift operations.

The detail of the dual-decimal-fuse technique are shown in Equations (12)–(14). At the base of Equation (4), we add bias (Bias) into the equation, because for each output channel of OFM of a convolutional layer, there is a corresponding bias. Consequently, we arrive at Equation (12).

$$\begin{aligned}S_3(Q_3-Z_3)&=\{\textstyle\sum S_1(Q_1[i]-Z_1)S_2(Q_2[i]-Z_2)+Bias\}\\&=S_1S_2\Big\{\textstyle\sum(Q_1[i]-Z_1)(Q_2[i]-Z_2)+\frac{Bias}{S_1S_2}\Big\}\end{aligned} \tag{12}$$

We can then obtain Equation (13).

$$(Q_3-Z_3)=\frac{S_1S_2}{S_3}\left\{\sum(Q_1[i]-Z_1)(Q_2[i]-Z_2)+\frac{Bias}{S_1S_2}\right\} \tag{13}$$

Because the convolution is 2-D, we add two $1/24$ of the Winograd kernel transformation matrix into S_1S_2/S_3 of Equation (13); we then derive M_{fuse} and Equation (14).

$$(Q_3-Z_3)=\frac{S_1S_2}{S_3\times24\times24}\left\{\sum(Q_1-Z_1)(Q_2-Z_2)\times24\times24+\frac{Bias}{S_1S_2}\times24\times24\right\} \tag{14}$$

In Equation (14), $\sum(Q_1-Z_1)(Q_2-Z_2)\times24\times24$ denotes that the Winograd kernel transformation matrix ($G_{6\times3}$) is multiplied by 24, and the new $G_{6\times3}$ is shown in Equation (15).

$$G_{6\times3}=\begin{bmatrix}6&0&0\\-4&-4&-4\\-4&4&-4\\1&2&4\\1&-2&4\\0&0&24\end{bmatrix} \tag{15}$$

Based on the fact that $24\times24=8\times3\times8\times3=64\times9=2^6\times9$, Equation (15) can be further transformed into Equation (16).

$$(Q_3-Z_3)=\frac{S_1S_2}{S_3\times9}\left\{\left[\sum(Q_1[i]-Z_1)(Q_2[i]-Z_2)\times24\times24\right]\gg6+\frac{Bias}{S_1S_2}\times9\right\} \tag{16}$$

In PyTorch, S_1S_2/S_3 can be transformed into multiply and shift operations (see Equation (6)). By using the same method, $S_1S_2/(S_3\times9)$ can also be transformed into multiply and shift operations. Therefore, Equation (16) can be optimized into Equation (17).

$$(Q_3-Z_3)=MULT\left\{\left[\sum(Q_1[i]-Z_1)(Q_2[i]-Z_2)\times24\times24\right]\gg6+\frac{Bias}{S_1S_2}\times9\right\}\gg SHIFT \tag{17}$$

The dual-decimal-fuse technique makes the Winograd kernel transformation matrix 24 times larger, and decimals such as 1/6, 1/12, and 1/24 are all transformed into integers, which maintains accuracy with negligible loss.

3.2. Winograd Decomposed-Part Reuse Technique

The original Winograd only supports the kernel whose size is 3×3 and whose stride is 1. Paper [34] proposes DWM, which makes Winograd support all types of kernels, and validates on GPU. When we implement DWM on FPGA, we must consider how to save LUT resources across different types of kernels. For example, Table 1 shows the transformation function of the original DWM and Winograd decomposed-part reuse (WDPR) under four types of kernels.

Table 1. Transformation function of DWM and WDPR under four types of kernels.

Kernel Size (K × K)	Stride (S)	Original DWM		WDPR	
		Part	**Transformation**	**Part**	**Transformation**
3×3	1	1	F(4×4, 3×3)	1	F(4×4, 3×3)
3×3	2	1	F(4×4, 2×2)	1	F(4×4, 3×3)
		2	F(4×4, 2×1)	2	F(4×4, 3×3)
		3	F(4×4, 1×2)	3	F(4×4, 3×3)
		4	F(4×4, 1×1)	4	F(4×4, 3×3)
5×5	1	1	F(4×4, 3×3)	1	F(4×4, 3×3)
		2	F(4×4, 3×2)	2	F(4×4, 3×3)
		3	F(4×4, 2×3)	3	F(4×4, 3×3)
		4	F(4×4, 2×2)	4	F(4×4, 3×3)
5×5	2	1	F(4×4, 3×3)	1	F(4×4, 3×3)
		2	F(4×4, 3×2)	2	F(4×4, 3×3)
		3	F(4×4, 2×3)	3	F(4×4, 3×3)
		4	F(4×4, 2×2)	4	F(4×4, 3×3)
Total			7		1

From Table 1, we find that the original DWM uses different transformation functions for different decomposed parts. Therefore, for four types of kernels, there is a total of seven transformation functions. If we implement seven transformation modules, they will use extensive logic resources. Furthermore, for different layers, the utilization rate is low. For example, if a convolution layer's kernel size is 3×3 and stride is 2, the accelerator uses F(4×4, 2×2), F(4×4, 2×1), F(4×4, 1×2) and F(4×4, 1×1) transformation modules. The accelerator does not use F(4×4, 3×3), F(4×4, 3×2) and F(4×4, 2×3). Therefore, we propose the Winograd decomposed-part reuse technique; for different types of kernels, each decomposed part is padded into the same shape. Therefore, four decomposed parts of IFM are all 6×6; four decomposed parts of Kernel are all 3×3. In this way, the output of the EWMM of each decomposed part has the same shape of 6×6. Hence, we can add the output of the EWMM of each decomposed part, then do one OFM inverse computation. There are two OFM inverse modules in Figure 2, because the accelerator at most generates $2 \times 4 \times 4$ OFMs at one clock for the pooling layer. For the original DWM, there are four OFM inverse modules for four decomposed parts, as the four outputs of the EWMM have a different shape. Therefore, the Winograd decomposed-part reuse technique can increase the utilization rate of transformation modules and decrease logic resources for different types of kernels, which decreases power.

3.3. The Architecture of the Accelerator

The architecture of the accelerator is shown in Figure 2. There are four main modules: Unified computing PE arrays, the OFM Generator, parameters and instructions memory and control logic.

The control logic reads and decodes instructions from Instruction BRAM, and controls the other three main modules. Instructions contain CNN architecture information, such as kernel size, stride, input channels, output channels and IFM size, etc.

The unified computing PE arrays the complete Winograd convolution and fully connect computation. The EWMM module consists of two lines; each line includes 144 PEs ($6 \times 6 \times 4 = 144$). For K = 3, S = 2; K = 5, S = 1; K = 5, S = 2, the line computes four decomposed parts of the EWMM. Two lines can compute two input channels of the EWMM or two 4×4 OFMs. For K = 3, S = 1, one line computes four input channels of the EWMM. In particular, the PE is implemented by DSP, and we use cascade the function of DSP to support the Winograd decomposed-part reuse technique. There are odd PEs and even PEs which are shown in Figure 6a. The functions of odd PEs and even PEs are defined by Equations (18) and (19), respectively.

$$PCOUT = A \times B \tag{18}$$

$$P = D \times B + PCIN \tag{19}$$

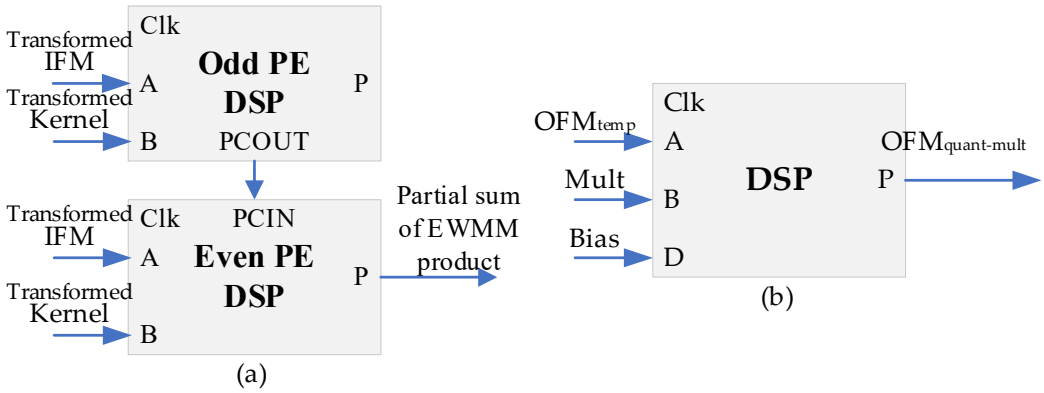

Figure 6. Multi-Mode DSP (**a**) DSP of PE. (**b**) DSP of Acc. & Quant. Multiply module.

The cascade function of the DSP computes the addition of the EWMM of the decomposed part, which avoids using LUT resources to implement Adder.

3.3.1. Ping-Pong BRAM with Multi-Mode BRAM

FPGA has many on-chip memory resources—BRAMs. We use multi-mode BRAM to store the activation value, which saves DFF resources of Configurable Logic Block (CLB) and decreases power. At first, the OFM is divided into several 4×4 parts (see Figure 7a), because the output size of Winograd F(4×4, 3×3) is 4×4. In the row dimension, there are odd R4s and even R4s. In the column dimension, there are odd C4s and even C4s. Therefore, there are four different colors, that is "Odd R4&Odd C4", "Even R4&Odd C4", "Odd R4&Even C4" and "Even R4&Even C4". In Figure 7b, each 4×4 part has a coordinator. In Figure 7c, there are four BRAMs to store four different colors of 4×4 parts. Four BRAMs works in a dual-port mode. By using four BRAMs to store the OFM, and using dual-port, Ping-pong BRAM can read $4 \times 4 \times 8$ IFMs at one clock, which meets the demand of large volumes of reading of IFM in the Winograd computation. By this method, we decrease the usage of DFF resources, which saves power, and increases reading speed.

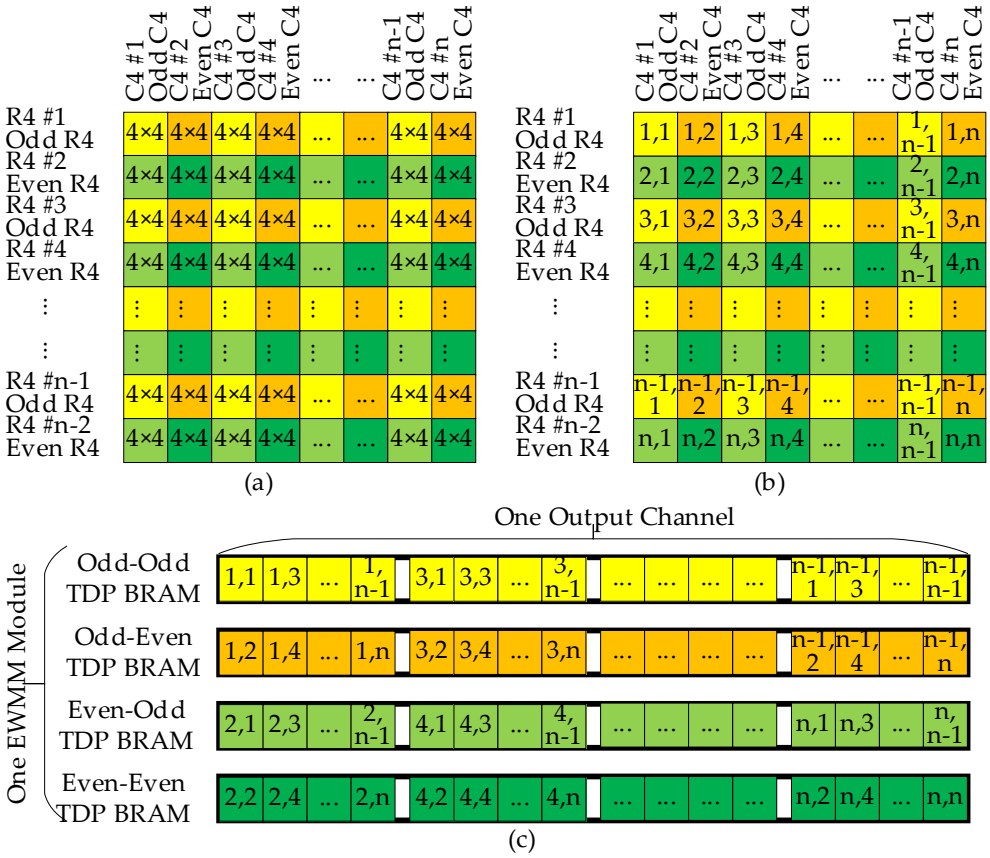

Figure 7. Ping-pong BRAM implemented by multi-mode BRAM: (**a**) OFM is divided into several 4 × 4 parts; (**b**) each 4 × 4 part has a coordinator; (**c**) Four BRAMs store all 4 × 4 parts of OFM.

3.3.2. Data Reuse and Padding of IFM Buffer

The accelerator supports four types of kernels, the IFM buffer reads the activation value from Ping-pong BRAM, which reuses data temporally at the column dimension, and reuses data spatially at the row dimension. It also supports padding, which preserves accuracy with negligible loss.

Figure 8a shows how the IFM buffer works when the kernel size is 5 × 5 and the stride is 1. Other types of kernels work similarly. The IFM buffer has eight ports to read activation values; that is, "Even R4 & Even C4 PortB", "Even R4 & Even C4 PortA", "Even R4 & Odd C4 PortB", "Even R4 & Odd C4 PortA", "Odd R4 & Even C4 PortB", "Odd R4 & Even C4 PortA", "Odd R4 & Odd C4 PortB" and "Odd R4 & Odd C4 PortA". When the kernel size is 5 × 5 and the stride is 1, "R1-R8 & C3-C10" is one output and "R5-R12 & C3-C10" is another output. Data of "R5-R8 & C3-C10" are reused spatially.

In Figure 8b, at clock 0, "R3-R12 & C13-C16" reads "1,1", "2,1" and "3,1" in Figure 7b. In Figure 8c, at clock 1, "R3-R12 & C13-C16" reads "1,3", "2,3" and "3,3" in Figure 7b; "R3-R12 & C9-C12" reads "1,2", "2,2" and "3,2" in Figure 7b. And "R3-R12 & C5-C8" reuses "1,1", "2,1" and "3,1". In Figure 8d, at clock 2, "R3-R12 & C1-C4" reuses "1,1", "2,1" and "3,1"; "R3-R12 & C5-C8" reuses "1,2", "2,2" and "3,2"; "R3-R12 & C9-C12" reuses "1,3", "2,3" and "3,3". Therefore, the IFM buffer reuses data temporally at the column dimension. In Figure 8c,d, at clock 1–2, the IFM buffer outputs 2 × 8 × 8, and supports "0" padding.

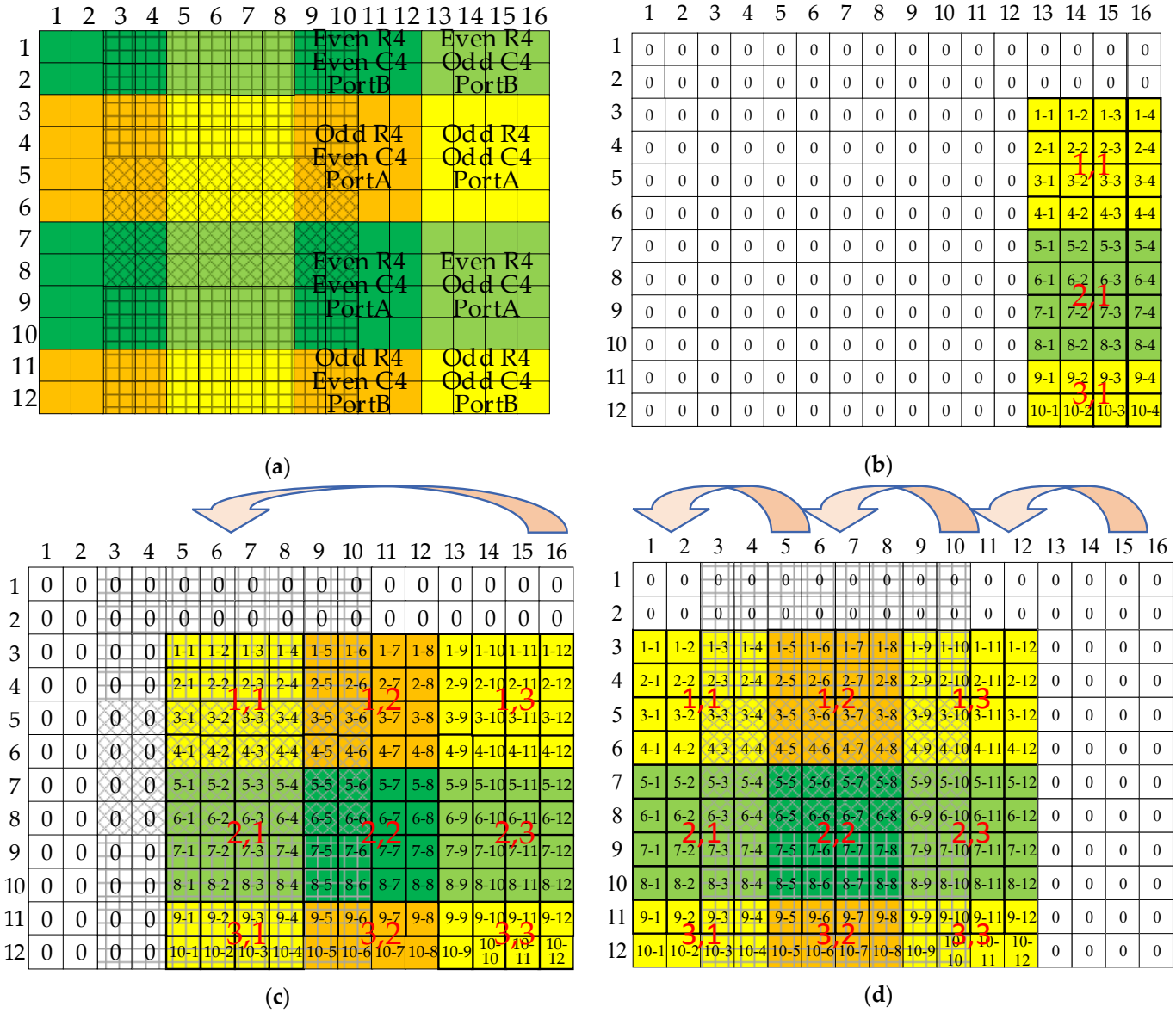

Figure 8. Data temporal & spatial reuse and padding of IFM buffer: (**a**) How IFM buffer works when kernel size is 5 × 5, stride is 1; (**b**) clock 0; (**c**) clock 1; (**d**) clock 2.

3.3.3. OFM Generator with Multi-Mode DSP

The details of the OFM generator are illustrated in Figure 9. The OFM generator uses multiply-accumulate, pre-adder and dynamic reconfiguration of the DSP.

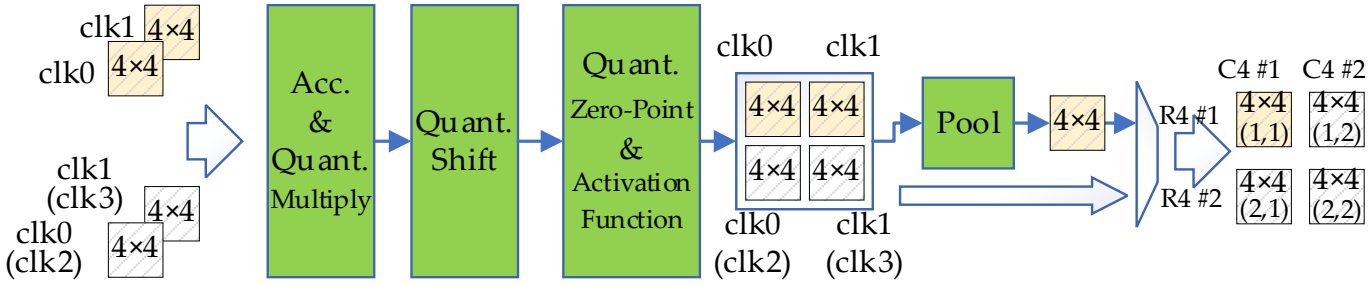

Figure 9. The details of OFM generator.

The partial input channel of IFMs generates temporary OFMs which are represented as OFM_{temp}, the OFM of all input channel of IFM is denoted as OFM_{acc}, and the bias is represented as $Bias$. Therefore, OFM_{acc} can be calculated by Equation (20). Equation (20) can be implemented by the accumulation function of the DSP.

$$OFM_{acc} = \sum_{1}^{Cin} OFM_{temp} + Bias \tag{20}$$

From Equation (7), the first step to quantize the OFM is multiplication by $MULT$. We then get $OFM_{quant-mult}$ in Equation (21).

$$OFM_{quant-mult} = \sum_{1}^{Cin} OFM_{temp} \times MULT + Bias \times MULT \tag{21}$$

The Acc. & Quant. multiply of Figure 9 completes the OFM accumulation and multiplication of quantization. It includes $4 \times 4 \times 4$ DSPs. The mode of the DSP is shown in Figure 6b. The DSP of Figure 6b implements two functions, which are defined by Equations (22) and (23).

$$P = A \times B + P \tag{22}$$

$$P = D \times B + P \tag{23}$$

Port A is OFM_{temp}, Port B is $MULT$, Port D is $Bias$, and Port P is $OFM_{quant-mult}$.

The OFM generator uses multiply-accumulate, pre-adder and dynamic reconfiguration of the DSP, which decreases usage of general programmable logic and saves power.

4. Experimental Results

We use our toolchain and accelerator to implement seven CNNs which support four types of kernels and more branches. When updating CNN, utilization of logic resources remains the same.

Experimental platform: Computer CPU: AMD Ryzen 7 5800H; DDR Frequency 3200 MHz; Xilinx VC709(XC7V690T, 28 nm), see Figure 10; Software: Vivado, PyTorch.

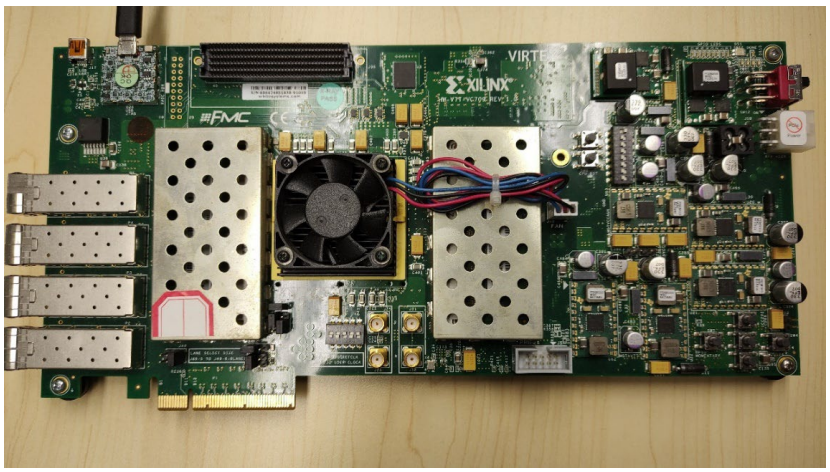

Figure 10. Xilinx VC709.

Experimental Process: On CIFAR-10 dataset, we use toolchain in Figure 1. The input of toolchain is the architecture of seven CNNs with four types of kernels. Four CNNs include branch, three CNNs do not include branch. We use PyTorch to train seven CNNs. For four CNNs with branch, we use inception-based branch-fuse technique to remove branch and keep main backbone, then use PyTorch to quantize the fused model. For three CNNs without branch, we directly use PyTorch to quantize the trained model.

The CNN updater of the toolchain generates seven groups of initialization file (.mem) of CNN parameters and instructions. CNN is defined by the content of BRAM. The content of BRAM is initialized by .mem files. We use one group of .mem files to generate the first bitstream by traditional design flow. We download the bitstream and check the image recognition result from LED of VC709. When updating CNN, we use a CNN updater to generate six other bitstreams, which eliminates the re-synthesis and re-implementation process. By this method we avoid a timing failure, unsuccessful routing and long development time. The differences between seven bitstreams are the content of BRAM which are CNN parameters and instructions. We download the bitstreams one by one and check the image recognition result from LED of VC709.

Table 2 shows seven CNNs be implemented on VC709. Papers [15,31] also use the toolchain to update CNN. Papers [15,31] both support three CNNs. The logic resources of paper [15,31] are changed when updating the CNN. Our accelerator keeps the logic resources unchanged.

Table 2. Seven CNNs are implemented.

NO.	Name	Kernel (Size, Stride) & Branch	TCAD 2020 [15]	TCAD 2020 [31]
1	VGG	K = 3, S = 1; No branches;	VGG	VGG
2	VGG-S2	K = 3, S = 1 & K = 3, S = 2; No branches;	-	-
3	AlexNet	K = 5, S = 1 & K = 3, S = 1; No branches;	-	AlexNet
4	RepResNet	K = 3, S = 1; With branches;	ResNet	ResNet
5	RepVGG	K = 3, S = 1 & K = 3, S = 2; With branches;	-	-
6	RepInception	K = 5, S = 1 & K = 3, S = 1; With branches;	Inception	-
7	RepK5S2	K = 5, S = 1 & K = 5, S = 2 & K = 3, S = 1 & K = 3, S = 2; With branches	-	-

Table 3 shows the accuracy of seven CNNs after training, quantization, and hardware accelerator. In Table 3, the largest accuracy loss between training and quantizing is −0.72% and all losses are within 1%. The largest accuracy loss between the accelerator and quantizing is −0.08% and all losses are within 0.1%, which indicates that our dual-decimal-fuse technique maintains accuracy with negligible loss.

Table 3. Accuracy of seven CNNs after training, quantization, and hardware accelerator.

CNN	After Training	After Quantizing	Loss between Training and Quantizing	Hardware Accelerator	Loss between Accelerator and Quantizing	Total Loss
VGG	93.82	93.75	−0.07	93.73	−0.02	−0.09
VGG-S2	92.6	92.59	−0.01	92.6	+0.01	0
AlexNet	90.72	90	−0.72	89.98	−0.02	−0.74
RepResNet	93.06	92.88	−0.18	92.93	+0.05	−0.13
RepVGG	92.99	92.93	−0.06	92.87	−0.06	−0.12
RepInception	93.56	93.36	−0.2	93.45	+0.09	−0.11
RepK5S2	93.29	93.14	−0.15	93.06	−0.08	−0.23

Table 4 shows comparison between FPGA-based CNN accelerator with toolchain. In Table 4, papers [14–16] include toolchain, and support CNN updating.

Table 4. Comparison between FPGA-based CNN accelerator with toolchain.

Items	**TCAD 2019 [14]**	**VLSI 2020 [16]**		**TCAD 2020 [15]**			**This Paper**		
>Platform	>VC709 (28 nm)	>XC7K325T		>Arria-10 (20 nm)			>VC709 (28 nm)		
>Toolchain	>Yes	>Yes		>Yes			>Yes		
Frequency (MHz)	150	200		240			100		
Precision	16-bit Fix-point	8-bit Fix-point		8/16 bit Fix-point	16-bit Fix-point	16-bit Fix-point	8/14/18 Integer		
Winograd	No	No		No			F(4 × 4, 3 × 3)		
Power	26	16.5		-			4.4		
Supported Kernels	K = 3, S = 1	K = 5, S = 1; K = 3, S = 1; K = 1, S = 1		K = 5, S = 1; K = 3, S = 1; K = 3, S = 2; K = 1, S = 1			K = 1, S = 1; K = 1, S = 2; K = 3, S = 1; K = 3, S = 2; K = 5, S = 1; K = 5, S = 2		
CNN	VGG	VGG16	InceptionV1	ResNet-50	VGG	Inception	RepResNet	VGG	RepInception
Logic Cell	300 K (81%)	94,763 (46.5%)	94,763 (46.5%)	286 K (67%)	228 K (49%)	277 K (65%)	255.6 K (59%)	255.6 K (59%)	255.6 K (59%)
BRAM(Kb)	1248 (42%)	165 (37.08%)	165 (37.08%)	2356 × 20 (87%)	2319 × 20 (85%)	1849 × 20 (68%)	1036 × 36 (70%)	1036 × 36 (70%)	1036 × 36 (70%)
DSP	2833 (78%)	516 (61.43%)	516 (61.43%)	3036 (100%)	3036 (100%)	3036 (100%)	2816 (78%)	2816 (78%)	2816 (78%)
Throughput (GOPS)	354	354	54.4	758	968.03	524.98	827.8	869.9	997.2
Power Efficiency (GOPS/W)	13.6	21.5	3.3	-	-	-	188.1	197.7	226.6

In Table 4, for inception, the throughput of paper [15] is 524.98, ours is 997.2 and is 1.9 times larger than paper [15]; the throughput of paper [16] is 54.4, ours is 997.2 and is 18.3 times larger than paper [16]. For ResNet, the throughput of paper [15] is 758, ours is 827.8 and is 1.1 times larger than paper [15]. The main reason is that our toolchain supports the inception-based branch–fuse technique. At the inference stage, it removes branches and keeps the main backbone, which increases throughput, while inception and ResNet of paper [15,16] don't have branch–fuse technique. At the inference stage, they have multi-branches, which consume more logic resources. The second reason is that our accelerator supports Winograd fast algorithm.

In Table 4, for the VGG, the throughput of paper [15] is 968.03, our is 869.9 and is smaller than paper [15]. Because the platform of paper [15] is Arria-10 (20 nm), which is more advanced than VC709 (28 nm), the frequency is 240 MHz while ours is 100 MHz. If our accelerator can work at 240 MHz, the throughput will larger than paper [15].

In Table 4, for the VGG, the throughput of paper [14] is 354, our is 869.9 and is 2.45 times larger than the paper [14]. The platform of paper [14] is VC709 (28 nm), which is the same as with our platform. DSP utilizations are both 78%. The frequency of paper [14] is 150 MHz, which is higher than ours (100 MHz). But our throughput is better because our accelerator uses Winograd fast algorithm, which uses fewer DSPs to compute convolution.

In Table 4, for the VGG, the throughput of paper [16] is 354, our is 869.9 and is 2.45 times larger than the paper [16], because our accelerator uses Winograd fast algorithm, and uses more logic resources.

Table 5 shows a comparison between the FPGA-based CNN accelerator with Winograd.

In Table 5, for the VGG, paper [32] and our accelerator all use VC709. The frequency of paper [32] is 150 MHz, which is higher than ours (100 MHz). The throughput of paper [32] is 570, our is 869.9 and is 1.5 times larger than the paper [32]. Because our accelerator uses Winograd F(4 × 4, 3 × 3), compared with Winograd F(2 × 2, 3 × 3) of paper [32], it uses fewer number DSPs to compute convolution.

Table 5. Comparison between FPGA-based CNN accelerator with Winograd.

Items	FPGA 2018 [32]	VLSI 2020 [30]		TCAD 2020 [31]			This Paper			
Platform	VC709 (28 nm)	Arria-10 (20 nm)		ZCU102 (16 nm)		ZC706 (28 nm)	VC709 (28 nm)			
Toolchain	No	No		Yes		Yes	Yes			
Frequency (MHz)	150	250		200		166	100			
Precision	16-bit Fix-point	16-bit Fix-point		16-bit Fix-point		16-bit Fix-point	8/14/18 Integer			
Winograd	F(2 × 2, 3 × 3)	F(2 × 2, 3 × 3) DWM		F(4 × 4, 3 × 3)			F(4 × 4, 3 × 3) DWM			
Power	25	18		-			4.4			
Supported Kernels	K = 3, S = 1	K = 3, S = 1; K = 3, S = 2		K = 5, S = 1; K = 3, S = 1; K = 3, S = 2; K = 1, S = 1			K = 1, S = 1; K = 1, S = 2; K = 3, S = 1; K = 3, S = 2; K = 5, S = 1; K = 5, S = 2			
CNN	VGG	VGG	VGG-S2	VGG	AlexNet	ResNet	VGG	VGG-S2	AlexNet	RepResNet
Logic Cell	175 K (40%)	181 K (15.7%)	180 K (15.7%)	95%	67%	67%	255.6 K (59%)	255.6 K (59%)	255.6 K (59%)	255.6 K (59%)
BRAM(Kb)	1232 (42%)	1310 (61.5%)	1310 (61.5%)	95%	67%	67%	1036 × 36 (70%)	1036 × 36 (70%)	1036 × 36 (70%)	1036 × 36 (70%)
DSP	1376 (38%)	1344 (88.5%)	1344 (88.5%)	95%	67%	67%	2816 (78%)	2816 (78%)	2816 (78%)	2816 (78%)
Throughput (GOPS)	570	1642	1788	2479.6	854.6	201.6	869.9	432.4	727.4	827.8
Power Efficiency (GOPS/W)	22.80	91.2	99.3	105.4	36.2	13.8	197.7	98.3	165.3	188.1

For VGG and VGG-S2, the reasons for the high GOPS in paper [30] are as follows: first, Arria-10 adopts 20 nm technology, which is more advanced than 28 nm technology of VC709; therefore, Arria-10 frequency can reach 250 MHz. Secondly, a DSP of Arria-10 contains two 19 × 18 multipliers in standard precision mode, which can achieve two 16 × 16 multiplication operations. Although 1344 DSPS are used, 1344 DSPS are equivalent to 2688 16 × 16 multipliers. A DSP of XC7V690T contains a 25 × 18 multiplier, which can only achieve a 14 × 18 multiplication operation. In our accelerator, 2304 DSPs complete the Winograd EWMM operation, and 512 DSPs complete the quantization operation. Therefore, the 2688 multipliers used in paper [30] exceed the 2304 multipliers used in our accelerator. In our accelerator, XC7V690T uses 28 nm and the clock frequency is 100 MHz. If the frequency of our accelerator can also reach 250 MHz, the GOPS of VGG can reach 2174.75 (=869.9 × 2.5), which is greater than 1642. The GOPS of the VGG-S2 can reach 1081 (=432.4 × 2.5), which is still lower than 1788, because our accelerator supports four types of kernels (K = 3, S = 1; K = 3, S = 2; K = 5, S = 1 and K = 5, S = 2). Paper [30] only supports two types of kernels (K = 3, S = 1 and K = 3, S = 2). The energy efficiency of VGG and VGG-S2 are 197.7 and 98.3, respectively. The energy efficiency of VGG is 2.17 times of that in paper [30], and the energy efficiency of VGG-S2 is 0.99 times of that in paper [30] because our accelerator uses the data reuse, makes full use of FPGA dedicated programmable resources, and reduces the use of FPGA general programmable resources and on-chip model to reduce power consumption.

For VGG, the reason for the high GOPS in paper [31] is that the accelerator only optimizes one network at a time and accelerates VGG with 95% of on-chip resources as the constraint of design space exploration. Moreover, ZCU102 adopts the 16 nm process, which is more advanced than the 28 nm process of VC709; therefore, the frequency of ZCU102 can reach 200 MHz. In addition, the logic cell used in paper [31] is 600 K × 0.95, which is 2.2 times that of 255.6 K in our accelerator. The clock frequency in our accelerator is

100 MHz, which is half of that in paper [31]. If the frequency of our accelerator can also reach 200 MHz, the GOPS for VGG can reach 1739.8 (=869.9 × 2), which is still lower than 2479.6. In paper [31], the weight conversion is converted in advance, and the converted weight is directly stored on FPGA. The weight transformation module of our accelerator is implemented on a chip, which requires real-time weight transformation. That leads to low GOPS. Finally, our accelerator supports four types of kernels (K = 3, S = 1; K = 3, S = 2; K = 5, S = 1 and K = 5, S = 2), while paper [31] supports one type of kernel (K = 3, S = 1). The energy efficiency of VGG is 1.87 times higher than that in paper [31] because our accelerator uses the data reuse, make full use of FPGA dedicated programmable resources, and reduces the use of FPGA general programmable resources and on-chip models to reduce power consumption.

For the AlexNet network, the accelerator in paper [31] optimizes only one network at a time. If the frequency of our accelerator can also reach 200 MHz, the GOPS can reach 1454.8 = (727.4 × 2), which is higher than 854.6, because the first layer of paper [31] uses standard convolution and does not use Winograd's algorithm, while our accelerator uses Winograd's algorithm with K = 5 and S = 1. Finally, our accelerator supports four types of kernels (K = 3, S = 1; K = 3, S = 2; K = 5, S = 1 and K = 5, S = 2), while paper [31] supports one type of kernel (K = 3, S = 1). Moreover, the energy efficiency of our accelerator is greater, and the energy efficiency of AlexNet is 4.57 times that of paper [31] because our accelerator uses the data reuse, makes full use of FPGA dedicated programmable resources, and reduces the use of FPGA general programmable resources and the on-chip model to reduce power consumption.

For the ResNet, the accelerator in paper [31] optimizes only one network at a time. The GOPS of ResNet in paper [31] is 201.6, and the GOPS of our accelerator is 827.8, which is 4.1 times better than that in paper [31]. The reasons for the low GOPS in paper [31] are as follows: first, the number of used DSPs is small, and even if all DSPs are used, there are only 900 DSPs. The second is that the 1 × 1 branch of ResNet consumes 40% of the resources. In our accelerator, we can use branch fusion technology to fuse the parameters of SKIP and 1 × 1 branches into the parameters of the 3 × 3 main backbone, and finally remove the SKIP and 1 × 1 branches, and keep only the 3 × 3 main backbone. The energy efficiency of our accelerator is 4.57 times higher than that of ResNet in paper [31].

5. Conclusions

We propose HBCA: a toolchain and corresponding FPGA-based accelerator to balance accuracy and speed when updating CNN. The toolchain proposes an inception-based branch–fused technique to balance accuracy and speed, which supports more branches and more types of kernels. The accelerator proposes a dual decimal–fused technique to balance accuracy and speed; the decimal of the Winograd transformation matrix is fused into the scale decimal of the 8-bit integer quantization, and all computation is transformed into integer computation.

The accelerator uses multi-mode BRAM to store the activation value, which saves DFF resources and decreases power. The accelerator uses multiply-cascade of the DSP to compute the EWMM and add the product of the EWMM. It also uses multiply-accumulate, pre-adder and dynamic reconfiguration of the DSP to generate and quantize the OFM. The multi-mode DSP saves LUT resources and decreases power. The accelerator supports four types of kernels, reuses IFM data temporally at the column dimension, and reuses data spatially at the row dimension. It also supports IFM padding, which maintains accuracy with negligible loss.

Experiments show that HBCA supports seven CNNs with four types of kernels and more branches. The accuracy loss of seven CNNs is within 0.1% compared to corresponding quantized models. For the inception, the throughput and power efficiency of our accelerator are 997.2 GOPS and 226.6 GOPS/W, respectively, which are higher than other FPGA-based CNN accelerators. For ResNet, the throughput and power efficiency of our accelerator are 827.8 GOPS and 188.1 GOPS/W, respectively, which are better than other FPGA-based

CNN accelerators. The main reason for the better performance of our accelerator is that our toolchain supports the inception-based branch–fuse technique. At the inference stage, it removes branches and keeps the main backbone, thereby increasing the throughput.

Benefiting from employing the Winograd fast algorithm, we utilized several DSPs to compute the convolution. In addition, our accelerator uses an on-chip model, multi-mode BRAM and DSP, the Winograd decomposed-part reuse technique and data-reuse. Thus, the power efficiency (GOPS/W) of VGG is up to 197.7, which is higher than other FPGA-based CNN accelerators. Although the power efficiency (GOPS/W) of VGG-S2 is 98.3, which is smaller than 99.3, our accelerator can support more types of kernels. Hence, our accelerator is more flexible.

In summary, HBCA enables efficient processing of CNNs to improve power efficiency and throughput without sacrificing accuracy or incurring additional hardware costs when updating CNN.

Author Contributions: Conceptualization, Z.L. and J.L.; methodology, Z.L., L.H. and J.W.; software, Z.L. and J.W.; validation, X.T. and J.L.; formal analysis, X.T., J.W. and J.L.; investigation, L.H. and J.L.; resources, X.T., J.W. and J.L.; data curation, Z.L. and L.H.; writing—original draft preparation, Z.L.; writing—review and editing, J.L.; visualization, Z.L. and L.H.; supervision, X.T., J.W. and J.L.; project administration, X.T., J.W. and J.L.; funding acquisition, X.T. and J.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China, grant number 62074101.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data are available upon request from the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kaiming, H.; Xiangyu, Z.; Shaoqing, R.; Jian, S. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
2. Elhassouny, A.; Smarandache, F. Trends in deep convolutional neural Networks architectures: A review. In Proceedings of the 2019 International Conference of Computer Science and Renewable Energies (ICCSRE), Agadir, Morocco, 22–24 July 2019; pp. 1–8. [CrossRef]
3. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *NIPS* **2017**, *60*, 84–90. [CrossRef]
4. Jie, H.; Li, S.; Samuel, A.; Gang, S.; Enhua, W. Squeeze-and-Excitation Networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 15–20 June 2019.
5. Ding, X.; Zhang, X.; Ma, N.; Han, J.; Ding, G.; Sun, J. RepVGG: Making VGG-style ConvNets Great Again. In Proceedings of the 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Nashville, TN, USA, 20–25 June 2021.
6. Liu, C.; Zoph, B.; Neumann, M.; Shlens, J.; Hua, W.; Li, L.; Fei-Fei, L.; Yuille, A.; Huang, J.; Murphy, K. Progressive neural architecture search. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 19–34.
7. Radosavovic, I.; Kosaraju, R.P.; Girshick, R.; He, K.; Dollar, P. Designing network design spaces. In Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 13–19 June 2020; pp. 10428–10436.
8. Freund, K. Machine Learning Application Landscape. 2017. Available online: https://www.xilinx.com/support/documentation/backgrounders/Machine-Learning-Application-Landscape.pdf (accessed on 28 February 2020).
9. Véstias, M.P.; Duarte, R.P.; De Sousa, J.T.; Neto, H.C. Moving Deep Learning to the Edge. *Algorithms* **2020**, *13*, 125. [CrossRef]
10. Zhang, Y.; Wei, X.-S.; Zhou, B.; Wu, J. Bag of Tricks for Long-Tailed Visual Recognition with Deep Convolutional Neural Networks. *Proc. Conf. AAAI Artif. Intell.* **2021**, *35*, 3447–3455. [CrossRef]
11. Grigorescu, S.; Trasnea, B.; Cocias, T.; Macesanu, G. A survey of deep learning techniques for autonomous driving. *J. Field Robot.* **2020**, *37*, 362–386. [CrossRef]
12. Wang, X.; Han, Y.; Leung, V.C.M.; Niyato, D.; Yan, X.; Chen, X. Convergence of Edge Computing and Deep Learning: A Comprehensive Survey. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 869–904. [CrossRef]
13. Véstias, M. A Survey of Convolutional Neural Networks on Edge with Reconfigurable Computing. *Algorithms* **2019**, *12*, 154. [CrossRef]

14. Zhang, C.; Sun, G.; Fang, Z.; Zhou, P.; Pan, P.; Cong, J. Caffeine: Towards Uniformed Representation and Acceleration for Deep Convolutional Neural Networks. *IEEE Trans. Comput. Des. Integr. Circuits Syst.* **2019**, *38*, 2072–2085. [CrossRef]
15. Ma, Y.; Cao, Y.; Vrudhula, S.; Seo, J.S. Automatic Compilation of Diverse CNNs Onto High-Performance FPGA Accelerators. *IEEE Trans. Comput. Des. Integr. Circuits Syst.* **2020**, *39*, 424–437. [CrossRef]
16. Yu, Y.; Wu, C.; Zhao, T.; Wang, K.; He, L. OPU: An FPGA-Based overlay processor for convolutional neural networks. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2019**, *28*, 35–47. [CrossRef]
17. Yu, Y.; Zhao, T.; Wang, K.; He, L. Light-OPU: An FPGA-based Overlay Processor for Lightweight Convolutional Neural Networks. In Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Seaside, CA, USA, 23–25 February 2020; pp. 122–132. [CrossRef]
18. Iandola, F.N.; Han, S.; Moskewicz, M.W.; Ashraf, K.; Dally, W.J.; Keutzer, K. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and<0.5 MB model size. *arXiv* **2016**, arXiv:1602.07360.
19. David, R.; Duke, J.; Jain, A.; Reddi, V.J.; Jeffries, N.; Li, J.; Kreeger, N.; Nappier, I.; Natraj, M.; Wang, T.; et al. TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems. *Proc. Mach. Learn. Syst.* **2020**, *3*, 800–811.
20. Lin, J.; Chen, W.M.; Lin, Y.; Gan, C.; Han, S. MCUNet: Tiny Deep Learning on IoT Devices. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 11711–11722.
21. Li, Z.; Gao, J.; Lai, J. HBDCA: A Toolchain for High-Accuracy BRAM-Defined CNN Accelerator on FPGA with Flexible Structure. *IEICE Trans. Inf. Syst.* **2021**, *E104.D*, 1724–1733. [CrossRef]
22. Sze, V.; Chen, Y.H.; Yang, T.J.; Emer, J.S. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proc. IEEE* **2017**, *105*, 2295–2329. [CrossRef]
23. Qiu, J.; Wang, J.; Yao, S.; Guo, K.; Li, B.; Zhou, E.; Yu, J.; Tang, T.; Xu, N.; Song, S.; et al. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network. In Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 21–23 February 2016; pp. 26–35. [CrossRef]
24. Guo, K.; Sui, L.; Qiu, J.; Yu, J.; Wang, J.; Yao, S.; Han, S.; Wang, Y.; Yang, H. Angel-Eye: A Complete Design Flow for Mapping CNN Onto Embedded FPGA. *IEEE Trans. Comput. Des. Integr. Circuits Syst.* **2018**, *37*, 35–47. [CrossRef]
25. Krishnamoorthi, R. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv* **2018**, arXiv:1806.08342v1.
26. Wu, H.; Judd, P.; Zhang, X.; Isaev, M.; Micikevicius, P. Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation. *arXiv* **2020**, arXiv:2004.09602.
27. Shaydyuk, N.K.; John, E.B. Semi-Streaming Architecture: A New Design Paradigm for CNN Implementation on FPGAs. *arXiv* **2020**, arXiv:2006.08759v1.
28. Suda, N.; Chandra, V.; Dasika, G.; Mohanty, A.; Ma, Y.; Vrudhula, S.; Seo, J.-s.; Cao, Y. Throughput-Optimized OpenCL-based FPGA Accelerator for Large-Scale Convolutional Neural Networks. In Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 21–23 February 2016; pp. 16–25.
29. Lavin, A.; Gray, S. Fast algorithms for convolutional neural networks. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 4013–4021.
30. Yepez, J.; Ko, S.-B. Stride 2 1-D, 2-D, and 3-D Winograd for Convolutional Neural Networks. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2020**, *28*, 853–863. [CrossRef]
31. Liang, Y.; Lu, L.; Xiao, Q.; Yan, S. Evaluating Fast Algorithms for Convolutional Neural Networks on FPGAs. *IEEE Trans. Comput. Des. Integr. Circuits Syst.* **2020**, *39*, 857–870. [CrossRef]
32. Shen, J.; Huang, Y.; Wang, Z.; Qiao, Y.; Wen, M.; Zhang, C. Towards a Uniform Template-based Architecture for Accelerating 2D and 3D CNNs on FPGA. In Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 25–27 February 2018; pp. 97–106. [CrossRef]
33. Ahmad, A.; Pasha, M.A. FFConv: An FPGA-based Accelerator for Fast Convolution Layers in Convolutional Neural Networks. *ACM Trans. Embed. Comput. Syst.* **2020**, *19*, 1–24. [CrossRef]
34. Huang, D.; Zhang, X.; Zhang, R.; Zhi, T.; He, D.; Guo, J.; Liu, C.; Guo, Q.; Du, Z.; Liu, S.; et al. DWM: A Decomposable Winograd Method for Convolution Acceleration. *Proc. Conf. AAAI Artif. Intell.* **2020**, *34*, 4174–4181. [CrossRef]
35. Huang, C.; Dong, X.; Li, Z.; Song, T.; Liu, Z.; Dong, L. Efficient Stride 2 Winograd Convolution Method Using Unified Transformation Matrices on FPGA. In Proceedings of the 2021 International Conference on Field-Programmable Technology (ICFPT), Auckland, New Zealand, 6–10 December 2021; pp. 1–9. [CrossRef]
36. Yu, J.; Hu, Y.; Ning, X.; Qiu, J.; Guo, K.; Wang, Y.; Yang, H. Instruction driven cross-layer CNN accelerator with Winograd transformation on FPGA. In Proceedings of the 2017 International Conference on Field Programmable Technology (ICFPT), Melbourne, VIC, Australia, 11–13 December 2017; pp. 227–230.
37. Ma, Y.; Cao, Y.; Vrudhula, S.; Seo, J.-S. Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 25–27 February 2018; pp. 45–54. [CrossRef]