Reconfigurable Convolutional Kernels for Neural Networks on FPGAs

Martin Hardieck, Martin Kumm, Konrad Möller, Peter Zipf
University of Kassel, Germany
{hardieck,kumm,konrad.moeller,zipf}@uni-kassel.de

ABSTRACT

Convolutional neural networks (CNNs) gained great success in machine learning applications and much attention was paid to their acceleration on field programmable gate arrays (FPGAs). The most demanding computational complexity of CNNs is found in the convolutional layers, which account for 90% of the total operations. The fact that parameters in convolutional layers do not change over a long time interval in weight stationary CNNs allows the use of reconfiguration to reduce the resource requirements. This work proposes several alternative reconfiguration schemes that significantly reduce the complexity of sum-of-products operations. The proposed direct configuration schemes provide the least resource requirements and fast reconfiguration times of 32 clock cycles but require additional memory for the pre-computed configurations. The proposed online reconfiguration scheme uses an online computation of the LUT contents to avoid this memory overhead. Finally, a scheme that duplicates the reconfigurable LUTs is proposed for which the reconfiguration time can be completely hidden in the computation time. Combined with a few online reconfiguration circuits, this provides the same configuration memory and configuration time as a conventional parallel kernel but offers large resource reductions of up to 80% of the LUTs.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; • **Hardware** → *Reconfigurable logic and FPGAs*; Reconfigurable logic applications;

KEYWORDS

Convolutional Neural Networks, FPGA, runtime reconfiguration

ACM Reference Format:
Martin Hardieck, Martin Kumm, Konrad Möller, Peter Zipf. 2019. Reconfigurable Convolutional Kernels, for Neural Networks on FPGAs. In *The 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '19), February 24–26, 2019, Seaside, CA, USA*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3289602.3293905

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
FPGA '19, February 24–26, 2019, Seaside, CA, USA
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6137-8/19/02…$15.00
https://doi.org/10.1145/3289602.3293905

1 INTRODUCTION

Deep convolutional neural networks (CNNs) are one of the most successful machine learning approaches providing state-of-the-art classification accuracy for object detection and classification. This high accuracy is achieved by making the network *deep*, i.e., using several hidden layers. This makes CNNs computationally very demanding. Due to the inherent parallelism of CNNs, parallel computation schemes like the single-instruction-multiple-data (SIMD) architecture of graphics processing units (GPUs) are well suited to provide a high performance for training and inference. However, it was shown in recent work that field programmable gate arrays (FPGAs) are far less power demanding and achieve a similar throughput compared to GPUs when performing inference [21]. Implementing CNNs on FPGAs attracted much attention in the last years [1, 3, 11, 14, 20, 21, 23, 24, 27, 29, 31, 34]. FPGAs allow large degrees of freedom concerning data formats and widths, memory interfacing and degree of parallelism which all have to be selected carefully. Regarding the data format it was shown that fixed point formats with relatively low word sizes are sufficient compared to the hardware-demanding floating point format [12, 13, 21, 30]. Word sizes between 4 and 8 bit already reach floating point accuracy while binarized neural networks (BNNs) [6, 20, 29] and ternary CNNs [30, 35] offer further resource reduction at the price of a reduced accuracy. Regarding the complexity, the highest arithmetic complexity can be found in convolutional layers. It was reported that 90% of the total operations are required in the convolutional layers of LeNet, AlexNet and VGG16 [21].

Most of the previous FPGA-based CNN implementations [1, 3, 11, 14, 21, 23, 24, 27, 31, 34], implement the required sum-of-products (SOP) operation in a parallel fashion. Typically, previous designs heavily use the embedded multipliers or digital signal processing (DSP) blocks of the FPGAs when implementing their kernels. However, kernel sizes and the number of parallel kernels are often selected based on the limited availability of DSP blocks. It was, e. g., stated in [31] that "DSP and on-chip block RAM are the two critical resource types". In addition, for the low word-size designs discussed above DSP blocks have a poor utilization. Recent work on reconfigurable constant multipliers (RCMs) showed that 1) RCMs use considerably fewer resources compared to logic-based multipliers and 2) the reconfiguration of the coefficients is possible within very few clock cycles [15, 17, 18, 22, 32, 33]. These RCMs are based on the look-up table (LUT)-based KCM scheme proposed by Chapman [4, 5] and use either random access memory (RAM) [32, 33], shift register LUTs (SRLs) [15] or configurable look-up tables (CFG-LUTs) [17, 18, 22] as reconfigurable LUT to be able to exchange the coefficient at run-time. The most recent RCMs based on CFG-LUTs provide a reconfiguration time of 32 clock cycles, which is very fast compared to a full or partial FPGA reconfiguration. As

typically large amounts of data are processed using the same coefficients (often thousands of convolutions, see Section 2) by using the weight-stationary scheme [28], this short reconfiguration time is often negligible. Hence, RCMs provide a good replacement for the DSP blocks inside the convolutional layers of CNN architectures. Either to allow the addition of more convolutional kernels implemented in logic or to replace DSP blocks which can be used to process the fully connected layers.

This work proposes different reconfigurable architectures for fixed point CNN kernels that combine and refine several advanced results from previous work. They all use a variant of Chapman's KCM technique [4, 5] combined with fast LUT-based reconfiguration [15, 17, 18, 22, 32, 33], pipelined compressor trees [2, 19] and faithful rounding [8] together with an optional online reconfiguration scheme [18, 33] which is tailored to the most recent CFGLUTs. With these different architectures, resource efficient, fast and accurate CNN kernels are possible which provide different alternatives regarding reconfiguration time and memory demands. A secondary contribution of this work is an automated and flexible tool flow. It is made available as part of the open-source code generator FloPoCo [7, 9]. This tool may be used by other researchers to generate customized hardware kernels that can be plugged into their CNN architectures. Finally, results regarding the resource utilization, performance, reconfiguration time and reconfiguration memory are presented for the proposed reconfigurable CNN kernels as well as for the straight forward convolutions used in previous work using common kernel and data word sizes. This provides a guideline to decide what performance can be expected when embedding these kernels into a given architecture. The proposed computation kernels can be directly placed into previous CNN architectures that exploit the parallelism of the convolutional kernel as a parallel SOP operation [1, 21, 27]. They can also be used as an alternative in design space exploration tools like [23]. Our work does explicitly *not* propose another CNN architecture but instead the proposed kernels can be plugged in to virtually any of the current CNN architectures for FPGAs providing reconfigurable LUTs.

In the following, some properties of popular CNNs are analyzed to motivate the use of run-time reconfiguration in the context of CNN kernels.

2 PROPERTIES OF POPULAR CONVOLUTIONAL NEURAL NETWORKS

The convolutional layers basically compute an SOP which is typically followed by an additional offset called *bias*. Often it contains a nonlinear activation function (e.g., rectified linear units, ReLU). We will focus in the following on the SOP part of the kernel. The SOP computes the convolution of the input feature map with the parameters of the kernel and produces an output feature map. There are typically several input and output feature maps which are called *channels*. Table 1 summarizes properties of the convolutional layers of popular CNNs to give an overview. The *kernel size N* describes the size of the convolutional core which is typically a square number. The column *pad* shows the number of pixels a kernel may overlap at the borders with the input feature map. *Stride* describes the distance between two consecutive computations of a kernel.

Table 1: Properties of the convolutional layers in popular CNNs, N: kernel size, P: pad, S: stride

Conv Layer	N	P	S	IF Chan.	IF Size	OF Chan.	Identical Ops (O)
AlexNet							
1	11×11	0	4	3	300	96	5,476
2	5×5	2	1	96	36	256	1,296
3	3×3	1	1	256	18	384	324
4	3×3	1	1	384	18	384	324
5	3×3	1	1	384	18	256	324
VGG16							
1	3×3	1	1	3	224	64	50,176
2	3×3	1	1	64	224	64	50,176
3	3×3	1	1	64	112	128	12,544
4	3×3	1	1	128	112	128	12,544
5	3×3	1	1	128	56	256	3,136
6–7	3×3	1	1	256	56	256	3,136
8	3×3	1	1	256	28	512	784
9–10	3×3	1	1	512	28	512	784
11–13	3×3	1	1	512	14	512	196
ResNet34							
1	7×7	0	2	3	224	64	12,100
2–7	3×3	1	1	64	56	64	3,136
8	3×3	0	2	64	56	128	784
9–15	3×3	1	1	128	28	128	784
16	3×3	0	2	128	28	256	196
17–27	3×3	1	1	256	14	256	196
28	3×3	0	2	256	14	512	49
29–33	3×3	1	1	512	7	512	49
DarkNet19							
1	3×3	0	1	3	256	32	64,516
2	3×3	2	1	32	128	64	16,900
3	3×3	4	1	64	64	128	4,900
4	1×1	5	1	128	64	64	5,476
5	3×3	6	1	64	64	128	5,476
6	3×3	8	1	128	32	256	2,116
7	1×1	9	1	256	32	128	2,500
8	3×3	10	1	128	32	256	2,500
9	3×3	12	1	256	16	512	1,444
10	1×1	13	1	512	16	256	1,764
11	3×3	14	1	256	16	512	1,764
12	1×1	15	1	512	16	256	2,116
13	3×3	16	1	256	16	512	2,116
14	3×3	18	1	512	8	1024	1,764
15	1×1	19	1	1024	8	512	2,116
16	3×3	20	1	512	8	1024	2,116
17	1×1	21	1	1024	8	512	2,500
18	3×3	22	1	512	8	1024	2,500
19	1×1	23	1	1024	8	1000	2,916

The last column of Table 1 shows the number of identical convolution operations that have to be performed for each input feature map channel, i.e., it shows the number of convolutions for which the coefficients do not change.

It can be observed from Table 1 that the number of identical convolution operations is pretty large which enables the use of run-time reconfiguration and motivates our work. In the following, the background methodologies are introduced on which the proposed reconfigurable CNN kernel is based.

3 BACKGROUND METHODOLOGIES

The proposed architectures compose advanced compressor tree techniques, LUT-based KCM and run-time reconfiguration of the LUTs. This combination is the base of our CNN core implementations and is introduced in the following.

3.1 Generic LUT-based Constant Multiplication

The main idea in the LUT-based KCM approach [4, 5] is to split the constant multiplication into several smaller ones which are bit-shifted and added to get the final product. The size of the small constant multiplications is chosen such that it directly fits the input size of an FPGA LUT tabulating all partial constant multiplier results. Consider a two's complement number x with B_i bits:

$$x = -2^{B_\mathrm{i}-1} x_{B_\mathrm{i}-1} + \sum_{b=0}^{B_\mathrm{i}-2} 2^b x_b \tag{1}$$

If this number is multiplied by a constant c_n with B_c bits, the resulting $B_\mathrm{c} \times B_\mathrm{i}$ multiplication can be divided into several smaller multiplications of size $B_\mathrm{c} \times L$ (assuming that B_i is divisible by L), by rearranging the partial sums

$$\begin{aligned}
\underbrace{c_n \cdot x}_{B_\mathrm{c}\times B_\mathrm{i}\text{ Mult.}} &= c_n \left(\sum_{b=0}^{B_\mathrm{i}-2} 2^b x_b - 2^{B_\mathrm{i}-1} x_{B_\mathrm{i}-1} \right) \\
&= \underbrace{c_n \sum_{b=0}^{L-1} 2^b x_b}_{B_\mathrm{c}\times L\text{ Mult.}} + 2^L \underbrace{c_n \sum_{b=0}^{L-1} 2^b x_{b+L}}_{B_\mathrm{c}\times L\text{ Mult.}} + \ldots \\
&\quad + 2^{(K-1)L} \underbrace{c_n \left(\sum_{b=0}^{L-2} 2^b x_{b+(K-1)L} - 2^{L-1} x_{KL-1} \right)}_{B_\mathrm{c}\times L\text{ Mult.}} .
\end{aligned} \tag{2}$$

Now, the constant multiplication is performed by using $K = \left\lceil \frac{B_\mathrm{i}}{L} \right\rceil$ smaller multiplications of size $B_\mathrm{c} \times L$, each computing one partial product (PP). A K-input addition is used to add up all PPs. In case the input word size B_i is not divisible by L, the input x has to be sign extended to the next larger word size such that $B_\mathrm{i}' = KL$ bits. Setting L to the input size of the FPGA LUT allows a direct mapping of the $B_\mathrm{c} \times L$ constant multiplier to the FPGA LUTs. The structure of the generic constant multiplier is shown in Figure 1. For a signed multiplication, the LUT computing the most significant bits (MSBs) (rightmost sum in (2)) has to provide a signed result, while all other LUTs compute unsigned PPs.

3.2 Compressor Trees

One important part in the constant multiplication is the summation of all the bit shifted partial products (Σ block in Figure 1). This becomes even more demanding when several constant multiplications have to be performed in an SOP operation. Compressor trees offer a very efficient solution for this task. They were originally designed for multipliers but have evolved for generic arithmetic on FPGAs in the last years. While traditional compressor trees are designed of full adders and half adders, they map poorly to the LUTs and carry chains of FPGAs. It was shown by Parandeh-Afshar et al. that so called generalized parallel counters (GPCs) perform much better

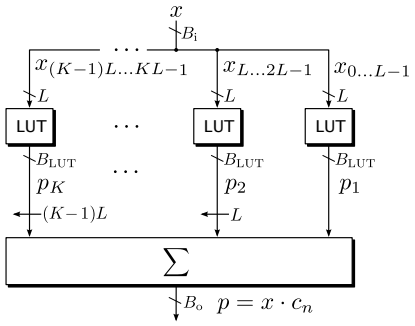

Figure 1: Generic LUT-based constant multiplier

on FPGAs [25, 26]. GPCs are multi-input compressors where each input may have a different weight which allows that the LUTs are efficiently utilized. Most recent work also utilizes the fast carry chains [16, 19, 26] which was a breakthrough in efficiency. The design of compressor trees is an optimization problem and several heuristics [2, 16, 26] as well as optimal approaches using integer linear programming [16, 19] are available. Current work shows that 40% of the LUT resources can be saved on average compared to common adder trees using 2-input adders and that the resulting circuits are 20% faster than adder trees built from ternary adders [16]. Therefore, compressor trees are used in this work for the partial product addition in KCM instead of using an adder tree.

3.3 Run-time Reconfigurable LUTs

Modern Xilinx FPGAs, namely Virtex 5/6, Spartan 6 and the whole 7 series (incl. Ultrascale and Ultrascale+), provide configurable look-up tables (CFGLUTs) whose content can be changed during runtime. They can be used as 5-input logic function with one output or as 4-input logic function with two outputs (one input less than a standard LUT) and provide a serial configuration interface consisting of the signals configuration data in (CDI), configuration data out (CDO), clock enable (CE), and configuration clock (CCLK). A change of the output function(s) can be performed by shifting 32 bit of new configuration data into CDI while CE is tied to a logical one. At the same time, the previous configuration is shifted out of CDO. This property can be used to cascade a series of CFGLUTs and to perform a serial reconfiguration of several CFGLUTs. However, each CDI input can be connected in parallel leading to a total reconfiguration time of 32 clock cycles for an arbitrary number of CFGLUTs. Alternatively to a CFGLUT, a shift-register LUT (for older Xilinx FPGA architectures) or block RAM (for other FPGAs) can be used as reconfigurable LUT. The latter is leading to some additional resources for an address counter and data multiplexers (MUXes) [33].

4 PROPOSED RUN-TIME RECONFIGURABLE CNN KERNEL ARCHITECTURE

We first introduce the generic architecture which is then further detailed in the following subsections.

4.1 Generic Architecture

The realization of a single LUT-based multiplication was shown in Section 3.1. However, a sum of products (SOP) is needed for the

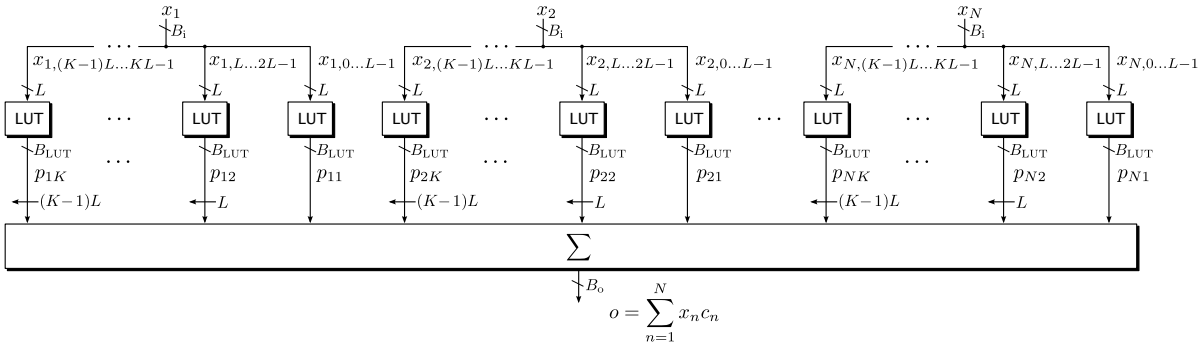

Figure 2: LUT-based SOP Architecture

convolutional kernel. Therefore, the proposed SOP architecture shown in Figure 2 is an application of the KCM approach and uses a compressor tree to sum up all partial products. The inputs $(x_1 \dots x_N)$ in Figure 2 all have the same word length B_i. Like before, each input x_n is split into K smaller parts of word length L. The size of L is equal to the number of inputs of the used LUT. The output word size of each partial product p_{nk} is thus $B_\mathrm{LUT} = L + B_\mathrm{c}$. Finally, all partial products are added with respect to the original position of their inputs. Therefore, each p_{nk} is left shifted with its corresponding weight (left arrows) before the summation using a compressor tree (Σ). However, as the full precision output word size B_o is much larger than the input word size, a truncation or rounding has to be performed. Instead of computing bits which are thrown away afterwards, we will consider in the following which parts of the partial product LUTs and compressor tree resources are actually necessary and will provide the bit level details of the architecture.

4.2 Reconfigurable SOP based on CFGLUTs

To allow the reconfiguration of weights, the configurable CFGLUTs are used. They are used in the 4-input mode providing two outputs (O5 and O6) as a previous analytic comparison revealed that this is always advantageous compared to a single 5-input LUT [18]. For that, the input I_4 has to be tied to a logical high. The inputs $I_0 \dots I_3$ of the CFGLUTs are connected to the N input signals which are each split into $L = 4$ bit chunks according to (2). Each block $(1 \dots K)$ processes a different 4-bit chunk of the input values of the N different kernel inputs $(x_1 \dots x_N)$. The resulting bits of the CFGLUT are added to the compressor tree with their corresponding shift.

The total number of different CFGLUTs contents to realize all partial products depends on the kernel size and the coefficient word size. An analytical upper bound for the number of partial product LUTs can be derived to be

$$\#\mathrm{LUT}_{\mathrm{PP,ub}} = 2N \left\lceil \frac{B_\mathrm{c} + 4}{2} \right\rceil . \tag{3}$$

4.3 Faithfully Rounded SOP

The output word size B_o of the SOP operation like shown in Figure 2 is much larger than the input word size B_i and has to be reduced. There are several possibilities to perform the word size reduction, of which truncation and rounding are among the most popular

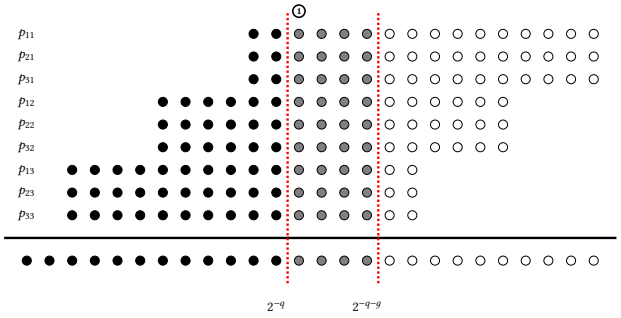

Figure 3: Example dot diagram for the proposed LUT-based SOP circuit for $N = 3$ coefficients, $B_\mathbf{i} = B_\mathbf{o} = B_\mathbf{c} = 12$ bit, $L = 4$

methods. Truncation to a precision of q MSBs (excluding sign) introduces a maximum error of 2^{-q} which is called one unit in the last place (ulp). The error of correct rounding is 2^{-q-1} ($\frac{1}{2}$ ulp). However, in both methods it is required that the full precision result is available.

To illustrate this, Figure 3 shows an example dot diagram of the compressor tree for a small SOP instance of $N = 3$ coefficients using word sizes for input, output and coefficients of $B_\mathrm{i} = B_\mathrm{o} = B_\mathrm{c} = 12$ bit. Each dot in the dot diagram represents one output bit of a LUT. The column of the dot (from right to left) corresponds to the weight of the bit. Adding all bits together leads to the SOP result which is indicated by the bits below the line. Even when only the upper q bits are considered in the preceding stage, all of these bits depend on results of the least significant columns due to carry propagation. This makes it necessary to compute the full result when using exact rounding or truncation.

However, the same accuracy as truncation can be achieved by using a scheme which is called *faithful rounding* [10], where the error is defined to be at most one ulp, i.e., 2^{-q}. It has the same error bound as truncation but does not necessarily lead to the same output vectors. This degree of freedom can be used to significantly reduce the hardware complexity by skipping the computation of intermediate results as long as the total error ϵ is less than or equal to one ulp, i.e., $\epsilon \leq 2^{-q}$.

A faithful rounding scheme of the SOP operation was proposed in [8] and is introduced next. First, observe that by simply removing all the dots that are located right of the dotted line at weight 2^{-q}

of Figure 3 would introduce an error of 2^{-q-1} times the column height of the bit heap (which is nine in the example of Figure 3) and is hence larger than the intended $\epsilon < 2^{-q}$. The idea from [8] is to add g additional guard bits to reduce this error. Hence, all bits to the right of the dotted line at weight 2^{-q-g} (shown in white) are discarded (and of course their corresponding LUTs). As we have N constant multipliers, each contributing

$$K = \left\lceil \frac{B_\mathrm{i}}{L} \right\rceil \tag{4}$$

partial products from the LUTs, the total error is

$$\epsilon = KN2^{-q-g-1} \, . \tag{5}$$

Therefore, by selecting a sufficiently large number of guard bits g, the error can be made as small as necessary. Bounding the error to $\epsilon \leq 2^{-q}$ leads to

$$KN2^{-q-g-1} \overset{!}{\leq} 2^{-q} \tag{6}$$

$$g \geq \log_2(KN) - 1 \, . \tag{7}$$

Thus, setting $g = \lceil \log_2(KN) \rceil - 1$ guarantees that the error is faithfully rounded. However, the intermediate result has g additional guard bits and has to be rounded to the output precision q. To avoid another adder for the rounding that adds the bit at position 2^{-q-1}, the identity

$$\mathrm{round}(x) = \left\lfloor x + \frac{1}{2} \right\rfloor \tag{8}$$

is used. With that, the rounding is obtained by adding a constant one bit with weight 2^{-p-1} and performing a truncation of the final result. This bit is shown at the top of column of weight 2^{-p-1} in Figure 3.

4.4 Memory Requirements

Let M_kern be the coefficient memory requirement of a single kernel, which is for a conventional kernel

$$M_\mathrm{kern,conv} = N \times B_\mathrm{c} \, . \tag{9}$$

For a reconfigurable kernel, the pre-computed LUT content has to be stored. As for each CFGLUT, 32 bit of configuration data is necessary, the memory requirements are

$$M_\mathrm{kern,direct} = 32 \times \#\mathrm{LUT_{PP}} \leq 64N \left\lceil \frac{B_\mathrm{c} + 4}{2} \right\rceil \, . \tag{10}$$

Note that the actual number of PP LUTs $\#\mathrm{LUT_{PP}}$ is less than the upper bound provided in (3) due to the LUT elimination provided in Section 4.3. The memory footprint is about 32 times the memory footprint of the conventional kernel. As this may not be applicable for many applications, an online computation of LUT content is proposed in the following. Using this online configuration, only the coefficients have to be stored instead of the full configuration data, which leads to the same memory requirements as for the conventional kernel implementation.

4.5 Online Computation of LUT Contents

To reduce the large memory footprint of the reconfigurable CNN kernel, an online configuration circuit is proposed which is based on a serial multiplier [22] but extended to the required two-output LUTs. In order to perform an online configuration, the LUT content has to be computed in reverse order as the last CFGLUT entry

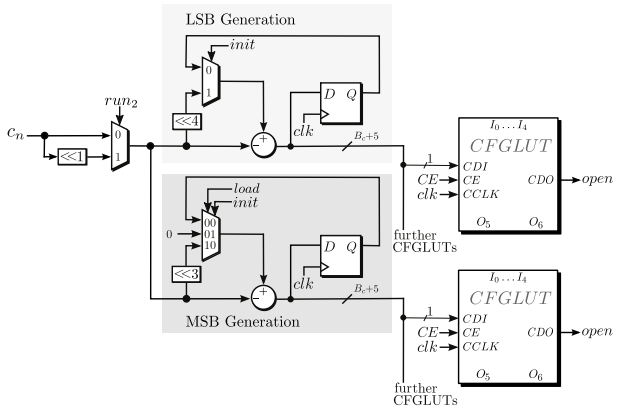

Figure 4: Proposed circuit to compute the LSB and MSB LUT content for online configuration

is shifted in first. Moreover, each CFGLUT is used as two output function and generates two bits of a partial product. Therefore, the lower part of the CFGLUT computes the bits with odd weights of the B_LUT bit result while the upper part of the CFGLUT computes the bits with even weight. As there is only one CDI input for lower and upper part we propose to first compute the odd bits and afterwards to compute the same but now left-shifted by one bit (which corresponds to a multiplication by two). In doing so, the lower and the upper parts of the table are computed one after the other. The proposed circuit to compute the LSB and MSB LUT contents is shown in Figure 4. The used control signals run_2 (r_2), *load* (l) and *init* (i), the general computation (*comp.*) as well as the LUT content for an example coefficient $c_n = 3$ are shown in Table 2 for each configuration step. Note that only the bits in bold face are shifted into CDI.

The inputs are the currently required coefficient c_n and the three control signals that control the MUXes. Signal run_2 is used to switch between the upper and lower part of the required CFGLUT by multiplying with two, while *init* and *load* are used to select the right minuend. Note that the results of the LSB generation on top and the MSB generation at the bottom provide a B_c+5 bit wide signal which is then split to provide a 1 bit wide CDI signal for several CFGLUTs. Each CFGLUT is connected to a global configuration clock and configuration enable. Thanks to the cascading possibility using CDI and CDO, the number of reconfiguration circuits R can be flexibly selected from $R = 1$ which requires a reconfiguration time of $32N$ clock cycles up to $R = N$ where only 32 cycles are required. Of course, at the cost of additional hardware resources.

4.6 Using Shadow LUTs to Hide the Reconfiguration Time

One important characteristic of the reconfiguration schemes described above is that the SOP circuit can not be used during reconfiguration. This is not a problem in the direct configuration scheme described in Section 4.2 as it only requires 32 clock cycles but may be prohibitive in the online configuration scheme above for a low number of reconfiguration circuits R. For that, a more efficient method can be used to effectively hide the reconfiguration

Table 2: Generated LUT contents for $L = 4$ and coefficient $c_n = 3$. Only the bits in bold face are shifted into CDI.

conf.	cntrl. sig.			MSB LUT		LSB LUTs (other LUTs)	
step	r_2	l	i	comp.	CDI signal	comp.	CDI signal
1	0	0	1	$-1 \times c_n$	**1** 1 **1** 1 1 **0** 1	$15 \times c_n$	**0** 0 **1** 0 **1** 1 **0** 1
2	0	0	0	$-2 \times c_n$	**1** 1 **1** 1 **1** 0 **1** 0	$14 \times c_n$	**0** 0 **1** 0 **1** 0 **1** 0
3	0	0	0	$-3 \times c_n$	**1** 1 **1** 1 **0** 1 **1** 1	$13 \times c_n$	**0** 0 **1** 0 **0** 1 **1** 1
4	0	0	0	$-4 \times c_n$	**1** 1 **1** 1 **0** 1 **0** 0	$12 \times c_n$	**0** 0 **1** 0 **0** 1 **0** 0
5	0	0	0	$-5 \times c_n$	**1** 1 **1** 1 **0** 0 **0** 1	$11 \times c_n$	**0** 0 **1** 0 **0** 0 **0** 1
6	0	0	0	$-6 \times c_n$	**1** 1 **1** 0 **1** 1 **1** 0	$10 \times c_n$	**0** 0 **0** 1 **1** 1 **1** 0
7	0	0	0	$-7 \times c_n$	**1** 1 **1** 0 **1** 0 **1** 1	$9 \times c_n$	**0** 0 **0** 1 **1** 0 **1** 1
8	0	0	0	$-8 \times c_n$	**1** 1 **1** 0 **1** 0 **0** 0	$8 \times c_n$	**0** 0 **0** 1 **1** 0 **0** 0
9	0	1	0	$7 \times c_n$	**0** 0 **0** 1 **0** 1 **0** 1	$7 \times c_n$	**0** 0 **0** 1 **0** 1 **0** 1
10	0	0	0	$6 \times c_n$	**0** 0 **0** 1 **0** 0 **1** 0	$6 \times c_n$	**0** 0 **0** 1 **0** 0 **1** 0
11	0	0	0	$5 \times c_n$	**0** 0 **0** 0 **1** 1 **1** 1	$5 \times c_n$	**0** 0 **0** 0 **1** 1 **1** 1
12	0	0	0	$4 \times c_n$	**0** 0 **0** 0 **1** 1 **0** 0	$4 \times c_n$	**0** 0 **0** 0 **1** 1 **0** 0
13	0	0	0	$3 \times c_n$	**0** 0 **0** 0 **1** 0 **0** 1	$3 \times c_n$	**0** 0 **0** 0 **1** 0 **0** 1
14	0	0	0	$2 \times c_n$	**0** 0 **0** 0 **0** 1 **1** 0	$2 \times c_n$	**0** 0 **0** 0 **0** 1 **1** 0
15	0	0	0	$1 \times c_n$	**0** 0 **0** 0 **0** 0 **1** 1	$1 \times c_n$	**0** 0 **0** 0 **0** 0 **1** 1
16	0	0	0	$0 \times c_n$	**0** 0 **0** 0 **0** 0 **0** 0	$0 \times c_n$	**0** 0 **0** 0 **0** 0 **0** 0
17	1	0	1	$-2 \times c_n$	**1** 1 **1** 1 **1** 0 **1** 0	$30 \times c_n$	**0** 1 **0** 1 **1** 0 **1** 0
18	1	0	0	$-4 \times c_n$	**1** 1 **1** 1 **0** 1 **0** 0	$28 \times c_n$	**0** 1 **0** 1 **0** 1 **0** 0
19	1	0	0	$-6 \times c_n$	**1** 1 **1** 0 **1** 1 **1** 0	$26 \times c_n$	**0** 1 **0** 0 **1** 1 **1** 0
20	1	0	0	$-8 \times c_n$	**1** 1 **1** 0 **1** 0 **0** 0	$24 \times c_n$	**0** 1 **0** 0 **1** 0 **0** 0
21	1	0	0	$-10 \times c_n$	**1** 1 **1** 0 **0** 0 **1** 0	$22 \times c_n$	**0** 1 **0** 0 **0** 0 **1** 0
22	1	0	0	$-12 \times c_n$	**1** 1 **0** 1 **1** 1 **0** 0	$20 \times c_n$	**0** 0 **1** 1 **1** 1 **0** 0
23	1	0	0	$-14 \times c_n$	**1** 1 **0** 1 **0** 1 **1** 0	$18 \times c_n$	**0** 0 **1** 1 **0** 1 **1** 0
24	1	0	0	$-16 \times c_n$	**1** 1 **0** 1 **0** 0 **0** 0	$16 \times c_n$	**0** 0 **1** 1 **0** 0 **0** 0
25	1	1	0	$14 \times c_n$	**0** 0 **1** 0 **1** 0 **1** 0	$14 \times c_n$	**0** 0 **1** 0 **1** 0 **1** 0
26	1	0	0	$12 \times c_n$	**0** 0 **1** 0 **0** 1 **0** 0	$12 \times c_n$	**0** 0 **1** 0 **0** 1 **0** 0
27	1	0	0	$10 \times c_n$	**0** 0 **0** 1 **1** 1 **1** 0	$10 \times c_n$	**0** 0 **0** 1 **1** 1 **1** 0
28	1	0	0	$8 \times c_n$	**0** 0 **0** 1 **1** 0 **0** 0	$8 \times c_n$	**0** 0 **0** 1 **1** 0 **0** 0
29	1	0	0	$6 \times c_n$	**0** 0 **0** 1 **0** 0 **1** 0	$6 \times c_n$	**0** 0 **0** 1 **0** 0 **1** 0
30	1	0	0	$4 \times c_n$	**0** 0 **0** 0 **1** 1 **0** 0	$4 \times c_n$	**0** 0 **0** 0 **1** 1 **0** 0
31	1	0	0	$2 \times c_n$	**0** 0 **0** 0 **0** 1 **1** 0	$2 \times c_n$	**0** 0 **0** 0 **0** 1 **1** 0
32	1	0	0	$0 \times c_n$	**0** 0 **0** 0 **0** 0 **0** 0	$0 \times c_n$	**0** 0 **0** 0 **0** 0 **0** 0

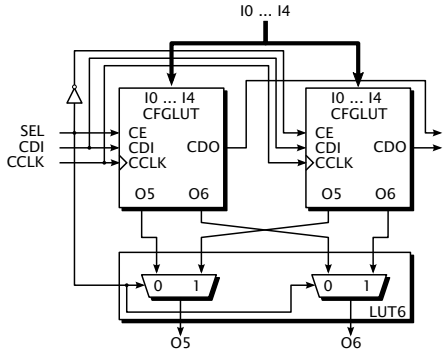

Figure 5: Reconfigurable LUT with *shadow LUT* for fast reconfiguration

time using a concept similar to *shadow registers*, where a set of registers is duplicated (typically doubled) and a control register selects the active register set. This allows to write data into the inactive (shadow) registers without immediate effects and to activate them in a single clock cycle. A similar concept is the *double buffering* scheme which is used to prevent visible artifacts in computer graphics. The same idea is used in this work by duplicating the CFGLUTs in the reconfigurable SOP circuit. We call this concept *shadow LUT* and its scheme is shown in Figure 5. The circuit in Figure 5 can replace a conventional CFGLUT. However, some control signals are different. Basically, a single CFGLUT is replaced by two CFGLUTs, one being the active (computing) LUT and the other being reconfigured. A third 5-input, 2-output LUT (another mode of the LUT6 in current Xilinx FPGAs) is used to implement two 2:1 multiplexers to select the active LUT. As long as the reconfiguration is faster than the computing time of convolution operations with identical coefficients, this completely hides the reconfiguration time as the reconfiguration can be performed during the computation and switching the select signal from one clock cycle to the other. The number of minimum reconfiguration units $R_{\min}$ to achieve this is simply the quotient between the reconfiguration time using one reconfiguration unit ($32N$) and the number of convolutions with identical coefficients O (see Table 1)

$$R_{\min} = \left\lceil \frac{32N}{O} \right\rceil . \tag{11}$$

Of course, the required LUTs to compute partial products are tripled (two for the CFGLUTs and one to implement the MUX) but due to the possible decrease in online reconfiguration units, this typically more than compensates the resources required for additional reconfiguration units. For the large nets, $R_{\min}$ is typically quite low as discussed in Section 5.3. Note that the conventional SOP also requires extra memory (e.g., shadow registers, distributed RAM or block RAM) to allow a reconfiguration within one clock cycle.

4.7 Properties of Different Reconfigurable Kernels

Table 3 summarizes the properties of the different architectures and their combination described above. Method "conventional SOP" corresponds to the standard implementation using N multipliers in parallel followed by an adder tree. It offers a low memory footprint for coefficient storage. Assuming that the coefficients are pre-loaded into registers (e. g., shadow registers) to allow a reconfiguration within one clock cycle, virtually no reconfiguration time ($T_{\text{rec}} = 0$) is necessary as no additional cycles are spent for reconfiguration. As will be proven in the results section, the conventional SOP will lead to a high resource usage.

Method "direct conf." corresponds to the architecture proposed in Section 4.2 in which the CFGLUTs are directly configured with their pre-computed LUT contents. This method requires the least resources and a fast reconfiguration of 32 clock cycles but naturally leads to the highest memory requirement which can be bounded to 32 times the requirements of the other methods. The concrete LUT requirements for $\#\text{LUT}_{\text{direct}}$, $\#\text{LUT}_{\text{shadow}}$, $\#\text{LUT}_{\text{rec}}$ and $\#\text{LUT}_{\text{ctrl}}$ are provided in the results section.

Method "direct conf., shadow LUTs" combines the direct configuration of Section 4.2 with the shadow LUTs proposed in Section 4.6. Compared to the direct configuration scheme, the reconfiguration time of 32 clock cycles is eliminated at the cost of additional resources.

Method "online conf." combines the direct configuration scheme of Section 4.2 with R reconfiguration units as discussed in Section 4.5. By selecting R, a tradeoff point between low resource demands but slow reconfiguration of $T_{\text{rec}} = 32N$ cycles by using $R = 1$ and a fast reconfiguration of $T_{\text{rec}} = 32$ cycles by using $R = N$ at the cost of a medium resource usage can be selected.

Finally, method "online conf., shadow LUTs" combines the online configuration scheme with shadow LUTs and online configuration.

Table 3: Comparison between conventional and our proposed reconfigurable kernels, $M_{\mathbf{kern}}$ denotes the memory requirements for parameters and $T_{\mathbf{rec}}$ is the reconfiguration time in clock cycles

Method	M_{kern}	T_{rec}	Resource Usage
conventional SOP	NB_{c}	0	high
direct conf.	$64N\lceil(B_{\text{c}}+4)/2\rceil \geq 32NB_{\text{c}}$	32	low (#LUT$_{\text{direct}}$)
direct conf., shadow LUTs	$64N\lceil(B_{\text{c}}+4)/2\rceil \geq 32NB_{\text{c}}$	0	medium (#LUT$_{\text{shadow}}$)
online conf.	NB_{c}	$32\left\lceil\frac{N}{R}\right\rceil$	low to medium (#LUT$_{\text{direct}}$ + $R\times$ #LUT$_{\text{rec}}$ + #LUT$_{\text{ctrl}}$)
online conf., shadow LUTs ($R=R_{\text{min}}$)	NB_{c}	0	medium (#LUT$_{\text{shadow}}$ + $R_{\text{min}}\times$ #LUT$_{\text{rec}}$ + #LUT$_{\text{ctrl}}$)

It is the most attractive scheme as the configuration time is completely hidden by using $R = R_{\text{min}}$ reconfiguration units, it has a low memory footprint and can be realized with very moderate LUT requirements.

5 EXPERIMENTAL RESULTS

5.1 Synthesis Results of Kernels

In order to quantify the resource usage and the maximum clock frequency, synthesis experiments have been performed for the proposed circuits. For that, an open-source VHDL code generator was implemented which is based on the FloPoCo [7, 9] library. This code generator is available in the "uni_ks" branch of the FloPoCo git repository [7]. It can be parametrized for different kernel sizes (N), different input, output and coefficient word lengths and the different reconfiguration schemes as discussed in the previous section.

For the evaluation, kernels with the most common sizes from 3×3 to 11×11 and different word lengths were generated. The word lengths of input, output and coefficients were set to be identical $B = B_{\text{i}} = B_{\text{c}} = B_{\text{o}}$, which was also assumed in most previous work. As lower word lengths are the most attractive, the word lengths between $B = 2\ldots12$ were evaluated for each kernel size. As reference, a code generator for the "conventional SOP" using N pipelined multipliers and an adder tree consisting of 2-input adders was implemented which should be close to the kernels used in previous work. The generated VHDL cores were synthesized for a Virtex 6 FPGA (XC6VLX760-FF1760-2) using Xilinx ISE 14.7.

The complete synthesis results after place&route for the conventional SOPs as well as the proposed SOPs are listed in Table 4. It shows the DSP and LUT resources, the maximum frequency (f_{max}) in MHz, the memory requirements of the kernel (M_{kern}) in bit as well as the latency in clock cycles. The conventional SOP was synthesized with two different synthesis options. In the "DSP-based" setting, the usage of DSP blocks was set to 'auto' while DSPs were prevented in the "logic only" setting. Plots of the LUT resources as well as the relative LUT reduction compared to the "logic only" reference are given in Figure 6. As our goal is to provide an alternative to the limited DSP blocks, the "logic only" results are used as a reference to enable fair comparisons.

The proposed methods use the same nomenclature as introduced in Section 4.7. The direct configuration schemes ("direct conf.") do not include any online reconfiguration circuits and are given with single CFGLUTs ($R = 0$) and the shadow CFGLUT scheme (shadow $R = 0$) of Section 4.6. Their LUT results directly correspond to the #LUT$_{\text{direct}}$ and #LUT$_{\text{shadow}}$ used in the comparison in Table 3. As the real memory M_{kern} footprint is typically less than the upper bound (10), the actual numbers are also tabulated in Table 4.

The methods using the online configuration ("online conf.") are computed by using the formulas given in Table 3. For that, the LUT results for the reconfiguration controller (#LUT$_{\text{rec}}$) are given for different coefficient word lengths in Table 5. The required LUTs for the control unit that generates the select signals were determined to be #LUT$_{\text{ctrl}} = 5$. The online configuration scheme with simple CFGLUTs was evaluated for the two extremes using $R = 1$ and $R = N$ reconfiguration units. The online configuration scheme with shadow LUTs was evaluated for the most common case $R = 1$ (see Section 5.3).

As can be observed from Figure 6, significant resource savings are possible by using the proposed reconfigurable circuits. The simplest direct configuration scheme without shadow LUTs achieves LUT reductions of 50% to 80% compared to the reference. Using the online configuration with one reconfiguration unit ($R = 1$) performs nearly identical with increasing kernel size due to the low overhead of a single reconfiguration unit. Using the other extreme of $R = N$ reconfiguration units shows the worst case where LUT reductions can be achieved for $B \geq 4$ bit, which are the most interesting word lengths for CNNs. Here, the online configuration with shadow LUTs requires much less resources and is also beneficial regarding configuration time (see Table 3). The obtained maximum frequencies are always above 100 MHz and lie often between 200 MHz and 300 MHz. Here, the kernel should not be the limiting factor in a larger CNN architecture.

5.2 Reconfiguration Time Overhead

To get an impression of the impact to CNN architectures, the relative time overhead which is defined as T_{rec}/O is provided in Table 6 for the layers having a different convolution operations count O. The direct configuration scheme and online configuration scheme with $R = N$ have both $T_{\text{rec}} = 32$ which corresponds to the fourth column while the online configuration scheme with $R = 1$ corresponds to the fifth column of Table 6. Of course, any R between 1 and N can be selected. It can be observed that the reconfiguration time overhead is low for the nets like AlexNet and VGG16 but increases towards the output layers as for those the input feature maps get smaller (see also Table 1). So, as a rule of thumb, the online configuration scheme with $R = 1$ fits best to the first layers, the R should be increased towards the output layers and an online configuration scheme with shadow LUTs should be chosen for the output layer(s) as it typically does not cause any time overhead.

Table 4: Synthesis results of conventional reference designs and several variants of the proposed design

Kernel size (N)	Word size (B)	conventional SOP							proposed												
		DSP-based			logic only				direct conf.					online conf.							
									R=0		shadow R=0			R=1		R=N		shadow R=1			
		DSPs	LUTs	f_{max}	LUTs	f_{max}	M_{kern}	Latency	LUTs	f_{max}	LUTs	f_{max}	M_{kern}	LUTs	f_{max}	LUTs	f_{max}	LUTs	f_{max}	M_{kern}	Latency
3×3	2	0	108	487	92	751	18	7	42	517	75	467	576	62	517	182	517	95	467	18	4
3×3	3	0	213	363	163	480	27	7	47	485	79	474	576	70	485	214	485	102	474	27	4
3×3	4	0	334	337	276	661	36	7	42	410	77	423	576	67	410	227	410	102	423	36	4
3×3	5	5	344	312	457	432	45	7	147	311	288	281	2304	175	311	359	311	316	281	45	5
3×3	6	5	428	298	572	437	54	7	140	306	240	249	1728	170	306	370	306	270	249	54	5
3×3	7	9	81	379	838	383	63	7	166	319	302	257	2304	199	319	423	319	335	257	63	5
3×3	8	9	91	309	864	382	72	7	142	301	250	256	1728	177	301	417	301	285	256	72	5
3×3	9	9	99	429	1174	373	81	7	277	271	561	262	3456	315	271	579	271	599	262	81	6
3×3	10	9	110	355	1360	325	90	7	284	318	477	259	2880	324	318	604	318	517	259	90	6
3×3	11	9	122	424	1607	372	99	7	309	301	559	253	3456	352	301	656	301	602	253	99	6
3×3	12	9	131	434	1844	345	108	7	281	317	489	235	2880	326	317	646	317	534	235	108	6
5×5	2	0	346	469	255	701	50	8	107	323	208	267	1600	127	323	487	323	228	267	50	5
5×5	3	0	614	347	462	474	75	8	131	327	211	282	1600	154	327	586	327	234	282	75	5
5×5	4	0	963	308	764	652	100	8	117	302	215	281	1600	142	302	622	302	240	281	100	5
5×5	5	13	1015	311	1242	434	125	8	396	284	794	243	6400	424	284	976	284	822	243	125	7
5×5	6	13	1256	295	1577	402	150	8	400	306	664	236	4800	430	306	1030	306	694	236	150	7
5×5	7	25	216	497	2291	376	175	8	438	290	831	261	6400	471	290	1143	290	864	261	175	7
5×5	8	25	242	409	2404	357	200	8	428	311	674	239	4800	463	311	1183	311	709	239	200	7
5×5	9	25	266	242	3253	364	225	8	808	238	1577	101	9600	846	238	1638	238	1615	101	225	8
5×5	10	25	293	342	3739	340	250	8	864	302	1609	225	9600	904	302	1744	302	1649	225	250	8
5×5	11	25	321	365	4420	348	275	8	909	251	1598	120	9600	952	251	1864	251	1641	120	275	8
5×5	12	25	346	368	5108	334	300	8	963	269	1620	228	9600	1008	269	1968	269	1665	228	300	8
7×7	2	0	665	441	519	671	98	9	227	323	400	267	3136	247	323	967	323	420	267	98	6
7×7	3	0	1205	311	904	438	147	9	241	325	435	282	3136	264	325	1128	325	458	282	147	6
7×7	4	0	1924	321	1503	602	196	9	220	305	435	252	3136	245	305	1205	305	460	252	196	6
7×7	5	25	2022	304	2459	420	245	9	775	271	1578	240	12544	803	271	1907	271	1606	240	245	7
7×7	6	25	2466	284	3131	419	294	9	673	279	1285	249	9408	703	279	1903	279	1315	249	294	7
7×7	7	49	420	435	4397	370	343	9	791	244	1609	256	12544	824	244	2168	244	1642	256	343	7
7×7	8	49	470	384	4647	366	392	9	791	290	1277	114	9408	826	290	2266	290	1312	114	392	7
7×7	9	49	518	309	6369	357	441	9	1603	294	3086	249	18816	1641	294	3225	294	3124	249	441	9
7×7	10	49	569	409	7365	354	490	9	1823	306	3118	229	18816	1863	306	3543	306	3158	229	490	9
7×7	11	49	621	308	8651	343	539	9	1754	303	3078	116	18816	1797	303	3621	303	3121	116	539	9
7×7	12	49	670	324	10001	334	588	9	1838	304	3133	241	18816	1883	304	3803	304	3178	241	588	9
9×9	2	0	1100	436	857	609	162	10	382	315	684	268	5184	402	315	1602	315	704	268	162	6
9×9	3	0	2025	333	1506	444	243	10	347	267	708	263	5184	370	267	1810	267	731	263	243	6
9×9	4	0	3198	318	2469	598	324	10	440	292	722	266	5184	465	292	2065	292	747	266	324	6
9×9	5	41	3363	306	4070	424	405	10	1428	290	2601	235	20736	1456	290	3296	290	2629	235	405	9
9×9	6	41	4157	285	5208	418	486	10	1260	295	2124	263	15552	1290	295	3290	295	2154	263	486	9
9×9	7	81	698	262	7367	375	567	10	1343	279	2621	262	20736	1376	279	3616	279	2654	262	567	9
9×9	8	81	780	308	7760	362	648	10	1369	272	2090	131	15552	1404	272	3804	272	2125	131	648	9
9×9	9	81	860	245	10528	307	729	10	2492	249	4879	115	31104	2530	249	5170	249	4917	115	729	10
9×9	10	81	943	330	12095	346	810	10	2520	287	4148	130	25920	2560	287	5360	287	4188	130	810	10
9×9	11	81	1031	369	14321	340	891	10	2802	303	4892	117	31104	2845	303	5885	303	4935	117	891	10
9×9	12	81	1112	345	16553	331	972	10	2524	273	4241	226	25920	2569	273	5769	273	4286	226	972	10
11×11	2	0	1487	439	1299	629	242	10	530	312	1007	262	7744	550	312	2350	312	1027	262	242	7
11×11	3	0	2982	328	2247	421	363	10	612	324	1050	256	7744	635	324	2795	324	1073	256	363	7
11×11	4	0	4699	310	3692	576	484	10	650	300	1058	240	7744	675	300	3075	300	1083	240	484	7
11×11	5	61	5036	305	6030	419	605	10	1942	294	3875	243	30976	1970	294	4730	294	3903	243	605	10
11×11	6	61	6218	288	7790	408	726	10	1920	304	3169	254	23232	1950	304	4950	304	3199	254	726	10
11×11	7	121	1030	343	10845	376	847	10	1999	290	3947	262	30976	2032	290	5392	290	3980	262	847	10
11×11	8	121	1152	309	11536	356	968	10	2036	296	3105	117	23232	2071	296	5671	296	3140	117	968	10
11×11	9	121	1272	260	15670	354	1089	10	3935	270	7490	116	46464	3973	270	7933	270	7528	116	1089	11
11×11	10	121	1395	264	18109	345	1210	10	4335	261	7595	219	46464	4375	261	8575	261	7635	219	1210	11
11×11	11	121	1519	275	21352	342	1331	10	4358	271	7512	128	46464	4401	271	8961	271	7555	128	1331	11
11×11	12	121	1640	263	24627	327	1452	10	4036	253	7648	219	46464	4081	253	8881	253	7693	219	1452	11

Table 5: Synthesis results of the online reconfiguration circuit of a single coefficient

Coefficient word length (B_c)	LUTs (#LUT_{rec})	f_{max}
2	15	796.8
3	18	728.9
4	20	769.8
5	23	708.7
6	25	708.7
7	28	727.3
8	30	690.6
9	33	702.3
10	35	670.7
11	38	685.4
12	40	685.4

5.3 Resource Reduction for the Online Configuration Scheme with Shadow LUTs

As the online configuration with shadow LUTs provides the same reconfiguration speed and memory footprint as the conventional implementation, a detailed case study for real CNNs is considered in the following. First, the minimal number of reconfiguration units was evaluated using (11) for the layers having a different convolution operations count O. The result is shown in column R_{min} in Table 6. However, for the much larger and challenging nets ResNet34 and DarkNet19 taking only one or two reconfiguration units are sufficient for all the layers but the last few where the feature maps become too small.

The last three columns of Table 6 show the LUT requirements of the reference kernel, the online configuration with shadow LUTs

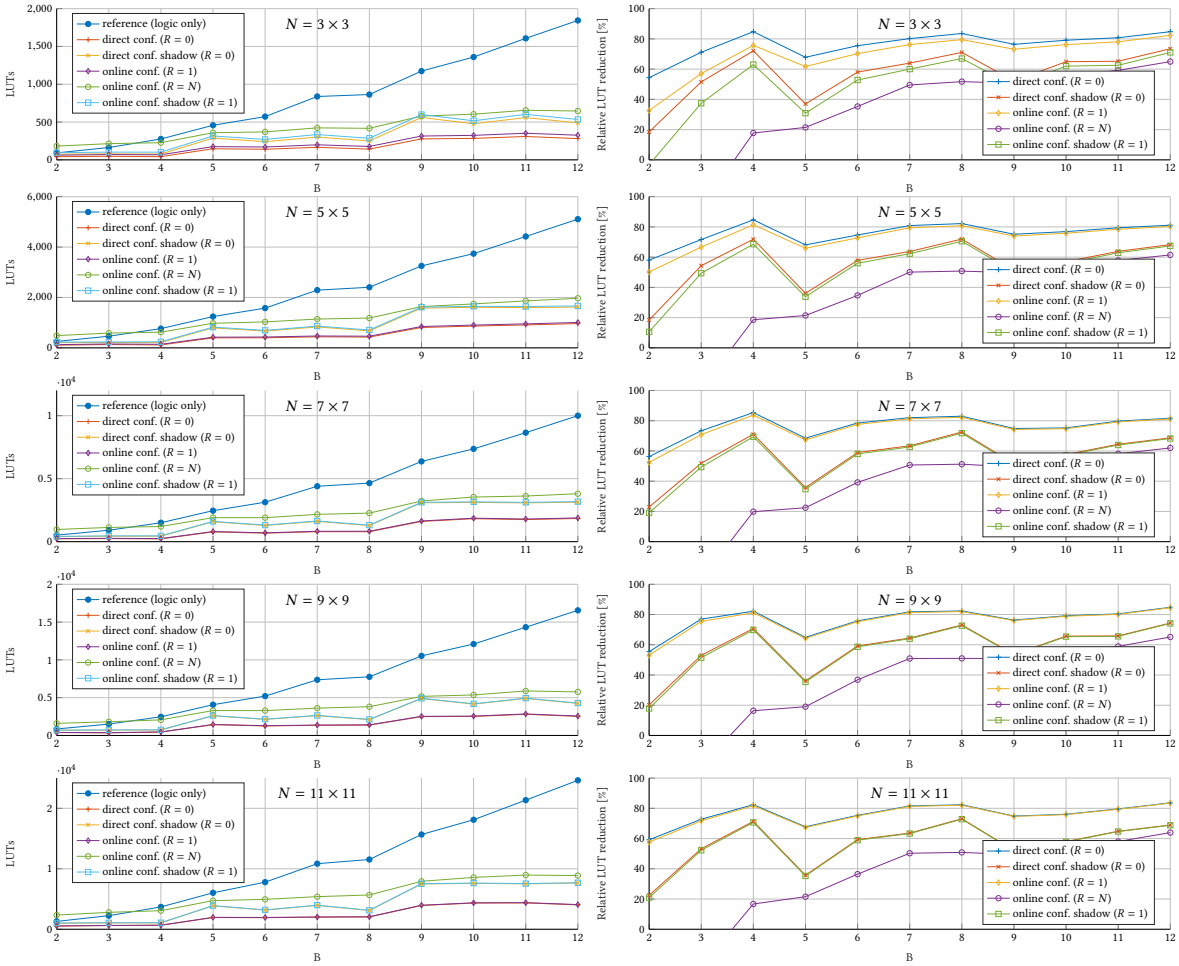

Figure 6: LUT results from synthesis (left) and the relative LUT reduction compared to the logic only reference (right) over the word size B for different kernel sizes N using the different methods

using $R_{\min}$ reconfiguration units and the corresponding LUT reduction for the common word size of $B = 8$ bit. It can be observed that 50 to 70% of the LUT requirements can be saved by using this reconfiguration scheme without any drawback concerning reconfiguration speed or memory footprint. Note that the reference in this comparison does not include the required buffers or shadow registers and corresponding multiplexers to enable this fast reconfiguration, so the actual resource reduction may be even higher.

6 CONCLUSION

Novel reconfigurable architectures were proposed to improve the sum-of-products operation used in convolutional kernels of CNNs. The alternative reconfiguration schemes were presented and analyzed regarding their memory footprint, reconfiguration time and resource usage. This analysis together with an experimental evaluation showed that the proposed architectures and schemes provide a tradeoff between these properties and offer LUT reductions of up to 80%. Even for the online configuration scheme with shadow LUTs, which offers the same memory footprint and reconfiguration time as a conventional parallel kernel implementation, LUT reductions in the order of 50% are obtained which makes this scheme an ideal candidate for future CNN architectures and should be considered in future design space explorations.

REFERENCES

[1] Marco Bettoni, Gianvito Urgese, Yuki Kobayashi, Enrico Macii, and Andrea Acquaviva. 2017. A Convolutional Neural Network Fully Implemented on FPGA for Embedded Platforms. In *New Generation of CAS (NGCAS)*. IEEE, 49–52.

[2] Nicolas Brunie, Florent de Dinechin, Matei Istoan, Guillaume Sergent, Kinga Illyes, and Bogdan Popa. 2013. Arithmetic Core Generation Using Bit Heaps. In *IEEE International Conference on Field Programmable Logic and Application (FPL)*. 1–8.

[3] Srimat Chakradhar, Murugan Sankaradas, Venkata Jakkula, and Srihari Cadambi. 2010. A Dynamically Configurable Coprocessor for Convolutional Neural Networks. *ACM SIGARCH Computer Architecture News* 38, 3 (June 2010), 247–257.

[4] Ken Chapman. 1996. Constant Coefficient Multipliers for the XC4000E. *Xilinx Application Note* (1996), 1–8.

[5] K D Chapman. 1994. Fast Integer Multipliers Fit in FPGAs. *Electronic Design News* (1994).

[6] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *arXiv.org* (Feb. 2016), 1–11. arXiv:cs.LG/1602.02830v3

[7] Florent de Dinechin. (accessed October 1, 2018). FloPoCo Project Website. http://flopoco.gforge.inria.fr

Table 6: Time overhead for reconfiguration schemes and LUT requirements for the kernels of convolutional layers in different nets for the conventional method and the online configuration with shadow LUTs

				Time Overhead Reconf. [%]		LUTs for $B = 8$ bit		
Layer	N	Identical Ops (O)	$R_{\min}$	$T_{\text{rec}} = 32$	$T_{\text{rec}} = 32N$	ref. (logic only)	online conf. shad. LUTs ($R = R_{\min}$)	LUT red.
AlexNet								
1	11×11	5,476	1	0.6	70.7	11536	3140	72.8%
2	5×5	1,296	1	2.5	61.7	2404	709	70.5%
3–5	3×3	324	1	9.9	88.9	864	285	67.0%
VGG16								
1–2	3×3	50,176	1	0.1	0.6	864	285	67.0%
3–4	3×3	12,544	1	0.3	2.3	864	285	67.0%
5–7	3×3	3,136	1	1.0	9.2	864	285	67.0%
8–10	3×3	784	1	4.1	36.7	864	285	67.0%
11–13	3×3	196	2	16.3	146.9	864	315	63.5%
ResNet34								
1	7×7	12,100	1	0.3	13.0	4647	1312	71.8%
2–7	3×3	3,136	1	1.0	9.2	864	285	67.0%
8–15	3×3	784	1	4.1	36.7	864	285	67.0%
16–27	3×3	196	2	16.3	146.9	864	315	63.5%
28–33	3×3	49	6	65.3	587.8	864	435	49.7%
DarkNet19								
1	3×3	64,516	1	0.0	0.4	864	285	67.0%
2	3×3	16,384	1	0.2	1.8	864	285	67.0%
3,5	3×3	4,096	1	0.8	7.0	864	285	67.0%
4	1×1	4,096	1	0.8	0.8	77	65	15.6%
6	3×3	1,024	1	3.1	28.1	864	285	67.0%
7	1×1	1,024	1	3.1	3.1	77	65	15.6%
8	3×3	1,024	1	3.1	28.1	864	285	67.0%
9,11,13	3×3	256	2	12.5	112.5	864	315	63.5%
10,12	1×1	256	1	12.5	12.5	77	65	15.6%
14,16,18	3×3	64	5	50.0	450.0	864	405	53.1%
15,17,19	1×1	64	1	50.0	50.0	77	65	15.6%

[8] Florent de Dinechin, Matei Istoan, and Abdelbassat Massouri. 2014. Sum-of-Product Architectures Computing Just Right. *IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP)* (2014), 41–47.

[9] F. de Dinechin and B. Pasca. 2012. Custom Arithmetic Datapath Design for FPGAs using the FloPoCo Core Generator. *IEEE Design & Test of Computers* 99 (2012), 1–6.

[10] T J Dekker. 1971. A Floating-Point Technique for Extending the Available Precision. *Numer. Math.* 18, 3 (June 1971), 224–242.

[11] Roberto DiCecco, Griffin Lacey, Jasmina Vasiljevic, Paul Chow, Graham Taylor, and Shawki Areibi. 2016. Caffeinated FPGAs: FPGA framework For Convolutional Neural Networks. In *2016 International Conference on Field-Programmable Technology (FPT)*. IEEE, 265–268.

[12] S Gupta, A Agrawal, K Gopalakrishnan Conference on Machine, and . 2015. Deep learning with limited numerical precision. *International Conference on Machine Learning* (2015), 1737–1746.

[13] Philipp Gysel. 2016. Ristretto: Hardware-Oriented Approximation of Convolutional Neural Networks. (May 2016), 1–63. arXiv:1605.06402

[14] Xushen Han, Dajiang Zhou, Shihao Wang, and Shinji Kimura. 2016. CNN-MERP: An FPGA-based memory-efficient reconfigurable processor for forward and backward propagation of convolutional neural networks. In *2016 IEEE 34th International Conference on Computer Design (ICCD)*. IEEE, 320–327.

[15] Javier Hormigo, Gabriel Caffarena, Juan P Oliver, and Eduardo Boemo. 2013. Self-Reconfigurable Constant Multiplier for FPGA. *ACM Transactions on Reconfigurable Technology and Systems* 6, 3 (Oct. 2013), 1–17.

[16] Martin Kumm and Johannes Kappauf. 2018. Advanced Compressor Tree Synthesis for FPGAs. *IEEE Trans. Comput.* 67, 8 (2018), 1078–1091.

[17] M Kumm, K Möller, and P Zipf. 2013. Dynamically Reconfigurable FIR Filter Architectures with Fast Reconfiguration. *International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)* (2013), 1–8.

[18] Martin Kumm, Konrad Möller, and Peter Zipf. 2013. Reconfigurable FIR Filter Using Distributed Arithmetic on FPGAs. In *IEEE International Symposium on Circuits and Systems (ISCAS)*. 2058–2061.

[19] Martin Kumm and Peter Zipf. 2014. Pipelined Compressor Tree Optimization Using Integer Linear Programming. In *IEEE International Conference on Field Programmable Logic and Application (FPL)*. IEEE, 1–8.

[20] Shuang Liang, Shouyi Yin, Leibo Liu, Wayne Luk, and Shaojun Wei. 2018. FP-BNN: Binarized Neural Network on FPGA. *Neurocomputing* 275 (2018), 1072–1086.

[21] Zhiqiang Liu, Yong Dou, Jingfei Jiang, Jinwei Xu, Shijie Li, Yongmei Zhou, and Yingnan Xu. 2017. Throughput-Optimized FPGA Accelerator for Deep Convolutional Neural Networks. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 10, 3 (July 2017), 17–23.

[22] Konrad Möller. 2017. *Run-time Reconfigurable Constant Multiplication on Field Programmable Gate Arrays*. Ph.D. Dissertation. Kassel University Press.

[23] Mohammad Motamedi, Philipp Gysel, Venkatesh Akella, and Soheil Ghiasi. 2016. Design space exploration of FPGA-based Deep Convolutional Neural Networks. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC*. IEEE, 575–580.

[24] Kalin Ovtcharov, Olatunji Ruwase, Joo-Young Kim, Jeremy Fowers, Karin Strauss, and Eric Chung. 2015. *Accelerating Deep Convolutional Neural Networks Using Specialized Hardware*. Technical Report. 1–4 pages.

[25] Hadi Parandeh-Afshar, Philip Brisk, and Paolo Ienne. 2008. Efficient Synthesis of Compressor Trees on FPGAs. In *Asia and South Pacific Design Automation Conference (ASPDAC)*. IEEE, 138–143.

[26] H. Parandeh-Afshar, Arkosnato Neogy, P. Brisk, and P. Ienne. 2011. Compressor Tree Synthesis on Commercial High-Performance FPGAs. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 4, 4 (Dec. 2011), 1–19.

[27] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang, and Huazhong Yang. 2016. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network. In *International Symposium on Field Programmable Gate Arrays (FPGA)*. ACM, 26–35.

[28] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. 2017. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proc. IEEE* 105, 12 (Dec. 2017), 2295–2329.

[29] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. 2017. FINN: A Framework for Fast, Scalable Binarized Neural Network Inference. In *International Symposium on Field-Programmable Gate Arrays (FPGA)*. ACM, New York, New York, USA, 65–74.

[30] G Venkatesh, E Nurvitadhi, and D Marr. 2017. Accelerating deep convolutional networks using low-precision and sparsity. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2861–2865.

[31] Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and Jason Cong. 2017. Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[32] Kazimierz Wiatr and Ernest Jamro. 2000. Constant coefficient multiplication in FPGA structures. In *Euromicro Workshop on Multimedia and Telecommunications*. IEEE Comput. Soc, 252–259.

[33] Kazimierz Wiatr and Ernest Jamro. 2001. Implementation of Multipliers in FPGA Structures. *International Symposium on Quality Electronic Design* (2001), 415–420.

[34] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. In *International Symposium on Field Programmable Gate Arrays (FPGA)*. ACM, 161–170.

[35] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. 2016. Trained Ternary Quantization. In *International Conference on Learning Representations*. 1–10.