PIR-DSP: An FPGA DSP block Architecture for Multi-Precision Deep Neural Networks

SeyedRamin Rasoulinezhad1, Hao Zhou2, Lingli Wang2 and Philip H.W. Leong1
1School of Electrical and Information Engineering, The University of Sydney, Australia 2006
2State Key Lab of ASIC and System, Fudan University, Shanghai 201203, China
Email: {seyedramin.rasoulinezhad,philip.leong}@sydney.edu.au
Email: {zhouhao,liwang}@fudan.edu.cn

Abstract—Quantisation is a key optimisation strategy to improve the performance of floating-point deep neural network (DNN) accelerators. Digital signal processing (DSP) blocks on field-programmable gate arrays are not efficiently utilised when the accelerator precision is much lower than the DSP precision. Through three modifications to Xilinx DSP48E2 DSP blocks, we address this issue for important computations in embedded DNN accelerators, namely the standard, depth-wise, and point-wise convolutional layers. First, we propose a flexible precision, run-time decomposable multiplier architecture for CNN implementations. Second, we propose a significant upgrade to DSP-DSP interconnect, providing a semi-2D low precision chaining capability which supports our low-precision multiplier. Finally, we improve data reuse via a register file which can also be configured as FIFO. Compared with the 27 × 18-bit mode in the Xilinx DSP48E2, our Precision, Interconnect, and Reuse-optimised DSP (PIR-DSP) offers a 6× improvement in multiply-accumulate operations per DSP in the 9 × 9-bit case, 12× for 4 × 4 bits, and 24× for 2 × 2 bits. We estimate that PIR-DSP decreases the run time energy to 31/19/13% of the original value in a 9/4/2-bit MobileNet-v2 DNN implementation.

I. INTRODUCTION

Recent progress with deep neural networks (DNNs) has yielded significant improvement over conventional approaches in cognitive applications like image, speech and video recognition [1]. Utilising massively parallel architectures, DNNs are much more memory and computationally expensive than previous approaches and efficient implementations continue to pose a challenge.

Modern CNN inference accelerators employ low precision arithmetic operations to decrease memory footprint and computation requirements [2], [3], [4], [5]. Reference [1] compared the implementation of multiply-accumulate (MAC) units with different wordlengths on Xilinx and Intel FPGAs. They reported that using fixed point 8×8-bit operations instead of single precision floating point, logic resources are reduced by 10 – 50×. This idea has been taken to its conclusion with ternary and binary operations which achieve extremely high speed and low energy on FPGA platforms [6], [7].

Current FPGAs include hard digital signal processing (DSP) blocks to allow efficient implementation of MAC operations. Unfortunately, as for central processing unit (CPU), graphics processing unit (GPU) and application specific integrated circuit (ASIC) architectures, they are optimised for higher precision (8-18 bits) and do not efficiently support low precision MAC operations, leading to inefficiencies in resource usage and energy consumption. Using high precision DSPs for low precision calculations is a waste of area and require additional LUT resources to implement the remaining operations if the DSPs are all utilised. In addition, researchers have proposed strategies involving run-time selection of wordlengths, which can not efficiently implemented in current FPGA architectures [8].

Research on computer architectures for DNN accelerators have extensively utilised 2D systolic architectures [9], [10]. Current FPGA DSP block layouts are based on 1D-DSP columns. This is a mismatch to 2D systolic architectures leading to inefficiencies and requiring that general purpose rather than dedicated routing resources be used.

To address the issues raised above, this paper proposes a novel precision, interconnect and reuse optimised DSP block, (PIR-DSP), which is optimised for implementing area-efficient DNNs. In particular, we make the following contributions:

- Precision: A parameterised MAC (MAC-IP) with runtime precision control, utilising a novel combination of chopping and recursive decomposition.
- Interconnect: A DSP interconnection scheme which provides support for semi-2D connections and low-precision streaming.
- Reuse: Inclusion of register files within the DSP to improve data-reuse and reduce energy.
- Evaluation of performance of the PIR-DSP, which incorporates the MAC-IP, interconnect and reuse optimisations, for implementing machine learning primitives including standard, depth-wise (DW) and point-wise (PW) convolution layers in recent embedded DNNs.

PIR-DSP is implemented as an open-source parameterised module generator which can target FPGAs or ASICs. All source code and data, along with a spreadsheet to reproduce all the results in this paper are available from http://github.com/raminrasoulinezhad/PIR-DSP.

II. BACKGROUND AND RELATED WORKS

In this section, we review recent embedded deep learning models and optimisation, systolic array solutions and their benefits, and FPGA DSP block architectures. A more thorough review is available in [15].
A. Embedded Deep Neural Networks

There has been considerable recent interest in memory and computationally efficient CNNs for mobile and embedded applications. Consider a standard convolutional layer which takes a $D_F \times D_F \times M$ feature map $F$ as input, and produces a $D_G \times D_G \times N$ feature map $G$ as output. The output is generated via a convolution with a $D_K \times D_K \times M \times N$ kernel $K$ as follows:

$$G_{k,l,n} = \sum_{i,j,m} K_{i,j,m,n} F_{k+i-1,l+j-1,m} \quad (1)$$

MobileNet [16], [12] proposed depth-wise separable convolutions which first factorises Equation 1 into $M$ depth-wise convolutions

$$\hat{G}_{k,l,m} = \sum_{i,j} \hat{K}_{i,j,m} F_{k+i-1,l+j-1,m} \quad (2)$$

where $\hat{K}$ is the $D_K \times D_K \times M$ depth-wise kernel and the $m$th filter of $\hat{K}$ is applied to the $m$th channel of $F$ to produce the $m$th channel of $\hat{G}$. Linear combinations of the $M$ depth-wise layer outputs are then used to form the $N$ outputs, these being called $1 \times 1$ point-wise convolutions. A speedup of $\frac{N+D_F^2}{N D_K^2}$ is achieved and typical values are $8 \times 9 \times$ (for $D_K = 3$), with a small reduction in accuracy. A study of the speed/accuracy tradeoffs of convolutional object detectors compared the use of the Inception, MobileNet, ResNet and VGG networks as the feature extractor for object detection, with MobileNet achieving excellent accuracy if low execution time on a GPU is desired [17].

In order to manage the massive computation and storage complexities of DNNs, efforts at reducing hardware resource usage at all design levels have been undertaken, e.g. efficient computational kernels [18], [19], [14], [20], [21], [22], data pruning [23], [24], memory compression [25], [26] and quantisation [27], [28], [29], [30], [31]. Table I provides a summary of the architectures employed in a number of recent state of the art embedded DNNs. From the last row, it can be seen that standard, DW, PW and fully connected (FC) layers account for almost all MACs.

Modern GPUs are presently the most popular solution for high-performance DNN implementation and Google’s Tensor Processing Unit (TPU) is an application specific integrated circuit (ASIC) for accelerating DNNs [32]. In contrast, FPGA architectures are more customisable and can support arbitrary precision MAC operations using fine-grained logic resources [6], [7], [33], [34], [35].

Interest in quantisation has dramatically increased since it was shown that binarised and ternary weights with low-precision activations, suffer only a small decrease in accuracy compared with floating point [36], [37]. Since FPGAs can implement arbitrary precision datapaths, they have some advantages over the byte addressable GPUs and CPUs for these applications. Moreover, the highest speed implementations on all platforms use reduced precision for efficiency reasons.

B. DSP blocks

CPU architectures working at high clock speeds and are efficient for highly sequential computations while GPU-based systems have a massive number of parallel processing elements and are favoured for parallel computations. In contrast to CPU and GPU architectures, FPGA systems are able to efficiently implement a range of parallel and sequential computations. They allow the data path to be better customised for an application, enabling designs to be more highly optimised, particularly in inference for processing single input feature maps (to minimise latency) and to support low precision.

1) Xilinx DSP48E2: The Xilinx DSP48E2 DSP [38] in the UltraScale architecture can perform $27 \times 18$ MAC operations and is illustrated in Figure 1. It includes a 27-bit pre-adder, 48-bit accumulator, and 48-bit arithmetic logic unit (ALU). In SIMD mode, dual 24-bit or quad 12-bit ADD/SUB operations can be computed in the ALU, and other DSP48E2 features

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-1/5 error</td>
<td>26% / 8.4%</td>
<td>28% / -</td>
<td>30.6% / -</td>
<td>42.5% / 19.7%</td>
</tr>
<tr>
<td># of CONV Stages</td>
<td>22</td>
<td>20</td>
<td>20</td>
<td>14</td>
</tr>
<tr>
<td># of standard conv. / Filter sizes</td>
<td>800 / 3</td>
<td>32 / 3</td>
<td>24 / 3</td>
<td>1376 / 3.7</td>
</tr>
<tr>
<td>standard conv. MAC / Parameter (%) Total</td>
<td>3.5% / 16.8%</td>
<td>3.4% / ~0%</td>
<td>5.7% / ~0%</td>
<td>72.1% / 45.5%</td>
</tr>
<tr>
<td># of DW Conv.s</td>
<td>15290</td>
<td>7136</td>
<td>2426</td>
<td>0</td>
</tr>
<tr>
<td>DW Conv. kernel / input / channel / Strides</td>
<td>3.5/7 / 7-57 / 11-176 / 1,2</td>
<td>3 / 3-112 / 32-960 / 1,2</td>
<td>3 / 7,14,28 / 24-232 / 1,2</td>
<td>- / - / - / -</td>
</tr>
<tr>
<td>DW Conv. MACs / Parameter (%) Total</td>
<td>14.1% / 5.4%</td>
<td>6.5% / 1.9%</td>
<td>2.7% / 1.0%</td>
<td>- / - / - / -</td>
</tr>
<tr>
<td># of PW Filters / Channel Depths</td>
<td>18465 / 11-1056</td>
<td>9920 / 16-960</td>
<td>5572 / 24-1024</td>
<td>2600 / 16-512</td>
</tr>
<tr>
<td>PW Conv. MACs / Parameters (%) Total</td>
<td>79.6% / 63.3%</td>
<td>88.7% / 61.2%</td>
<td>89.1% / 53.7%</td>
<td>24.5% / 54.5%</td>
</tr>
<tr>
<td>Global Pool Size</td>
<td>3x3</td>
<td>7x7</td>
<td>7x7</td>
<td>13x13</td>
</tr>
<tr>
<td>FC MACs / Parameters</td>
<td>~0.8M / ~0.8M</td>
<td>1.3M / 1.3M</td>
<td>1M / 1M</td>
<td>- / -</td>
</tr>
<tr>
<td>standard+PW+DW+FC MACs (%)</td>
<td>97.63</td>
<td>99.03</td>
<td>98.28</td>
<td>96.68</td>
</tr>
<tr>
<td>Total MACs / Parameters</td>
<td>564M / 5.3M</td>
<td>300M / 3.5M</td>
<td>141M / 2.3M</td>
<td>833M / 1.25M</td>
</tr>
</tbody>
</table>
include pattern matching and 1D unidirectional chaining connections. The DSPs can be cascaded to form a higher precision multiplier, and optional pipeline registers are present. In the DSP48E2, the SIMD mode wordlength can be changed at run-time.

2) Intel DSPs: Recent Intel DSPs [39] support one 27×27 or two 18×18 multiplications. Precision is compile-time rather than run-time configurable and there is no pattern matching unit. A pre-adder is implemented as well as two read-only register files (RFs) which can be initialised at compile-time and jointly operated as a higher precision RF. It is interesting that the predecessors were more flexible, Stratix IV supporting one 36-bit multiplication and up to eight 9×9 multiplications [40].

3) Previous Work in Multi-precision DSPs: Previous research has been conducted in supporting larger numbers of low precision operations using existing DSP blocks. Xilinx has proposed a method to use 8 DSP blocks to perform 7×2 8-bit multiply-add operations, achieving a 1.75× performance improvement over a naive implementation [41]. Colangelo et al. [42] proposed an 18×18 multiplier as four different 2×2 multipliers. Multi-precision FPGA hard blocks have been proposed by Parandeh-Afshar and Ienne [43]. This DSP variant, based on a radix-4 Booth architecture, supports 9/12/18/24/36 multiplier wordlengths and multi-input addition. Boutros et al. [44] proposed a modification of the Arria-10 DSP that can support 4×9-bit or 8×4-bit MACs. For the AlexNet, VGG-16, and ResNet-50 DNNs, this architecture improved speed by up to 1.6× while reducing utilised area by up to 30%.

The proposed PIR-DSP differs from previous designs in that it is a parameterised DSP block generator with improved flexibility, considers buffering of within the DSP, and also considers inter-DSP interconnect. This serves to improve the speed and energy consumption of the standard, DW and PW convolutions of Table I, with FC layer computations unaffected by our changes.

III. PIR-DSP: Architectural Modifications to the Xilinx DSP48E2 DSP Block

We now present our three modifications to the Xilinx DSP48E2 block (Figure 2).

A. Precision: Decomposable Multiplier

Our multiplier decomposition strategy is based on two approaches: chopping and recursive decomposition.

1) Chopping: A signed 2's complement number can be represented as the sum of one signed (the most significant part) and an unsigned term

\[ A^s = [a_{n-1}a_{n-2}...a_{k+1}]_2 \times 2^k + \sum_{j=0}^{k} [a_{k+j-1}a_{k+j}]_2 \]

(3)

where the \( k \)th bit is the dividing point and the \( A^s_H \) and \( A^s_L \) are the signed and unsigned portions. When applied to signed multiplication, this enables the separation of lower-precision product terms

\[ A^s B = A^s_H B^s_H 2^{2k} + A^s_H B^u_l 2^k + A^u_H B^s_L 2^k + A^u_H B^u_L 2^k \]

(4)

with each input being chopped at the \( k \)th bit.

Consider Equation 4 applied to an \( N \times M \)-bit multiplier with chopping size \( C \), where \( N \), \( M \), and \( C \) are respectively 27, 18, and 9. As shown in Figure 3(a), standard multiplication is done by summing the six partial results with appropriate shifts. Figure 3(b) shows that by controlling the shift steps for the first, fourth and fifth partial results, the summation can be arranged into two separate columns, where each column calculates a 3 \( C \times C \)-bit-MAC operation with separated carry-in signals

\[
\begin{align*}
\text{Out}_{\text{LSB}} &= P_0 + P_1 + P_2 + C_{in0} \\
\text{Out}_{\text{MSB}} &= P_3 + P_4 + P_5 + C_{in1}.
\end{align*}
\]

(5)

2) Recursive Decomposition: We employ the twin-precision technique [45] in a signed/unsigned \( N \times M \) multiplier. Inputs are 1-bit extended according to the individual sign control signals and their most significant bits (MSBs). The extended inputs are then multiplied using a \( (N+1) \times (N+1) \) signed multiplier based on the Baugh-Wooley structure [46]. Figure 4(a) shows the baseline multiplier where \( A \) and \( B \) are 9-bit numbers and each colored circle represents a logical function. By modifying the logic circuits of the PPs and preventing carry propagation using mode control signals, the multiplier can also operate as two half-precision multipliers. The required modifications are depicted in Figure 4(b). Figure 4(c) shows a recursive application of the technique to compute four quarter-precision values in parallel, only small changes to the PP logic and carry propagation paths being required.

Our multiplier is parameterised by chopping factors (separately for each of the two inputs) and the depth. For an \( M \times N \) multiplier, we use the notation \( M \times NC_{ij}DK \) where \( i \) and \( j \) are chopping factors (the numbers of times we chop \( M \) and \( N \)), and \( k \) is the recursive decomposition depth factor.

We applied our idea to the Xilinx DSP48E2 27×18 multiplier which produces two partial results (the following ALU is responsible for adding these two outputs). To create a 27×18C32D2 configuration, we chop \( A \) and \( B \) into \( i = 3 \) and \( j = 2 \) 9-bit parts. As each smaller multiplication is a signed/unsigned 9-bit multiplication, we then used recursive
decomposition with depth \( k = 2 \) to change the \( 9 \times 9 \) signed/unsigned multiplier to additionally support two \( 4 \times 4 \) or four \( 2 \times 2 \) multiplications (Figure 4(c)). Extra bits are included so that this is done without precision loss. Figures 3 (c) and (d) show how the bit-level carry propagation from each column to the next is arranged. Combining the six \( 9 \times 9 \) multipliers, we can compute the following multi-precision MAC operations without precision loss:

- One signed/unsigned \( 27 \times 18 \)
- Two sets of signed/unsigned \((9 \times 9 + 9 \times 9 \times 9 + 9 \times 9)\)
- Four sets of signed/unsigned \((4 \times 4 + 4 \times 4 + 4 \times 4 \times 4)\)
- Eight sets of signed/unsigned \((2 \times 2 + 2 \times 2 + 2 \times 2)\)

We have developed an IP generator which uses these techniques to convert any size multiplier to a MAC-IP. A sign-magnitude format is used so each operand can be signed or unsigned, this being controllable at run-time.

**B. Interconnect: Low-precision, Semi-2D DSP-DSP Communication**

Low energy and high performance DNN accelerators have been demonstrated using systolic array architectures [9], [10]. In this section, we focus on data movement among processing elements (PEs), which are DSP blocks in this content. In particular, \( 3 \times 3 \) convolutions are of most interest as these dominate the embedded DNNs reviewed in Section II.

Whereas in ASIC designs the PEs can be arranged in a 2D pattern, FPGA DSP blocks must be arranged in columns. In each column, DSP inputs and outputs can be passed via dedicated chain connections. This single-direction chaining is highly efficient for their intended signal processing applications. Although general routing resources make it possible to configure a 2D mesh network of PEs, this approach introduces significant amounts of additional circuitry and latency compared with direct connections.

In 2D systolic architectures, PE interconnections must forward input and result data to two different destination PEs, usually in different dimensions. Figure 5(a) shows a 2D PE architecture, proposed in [9], which is a \( N \times M \) mesh network of PEs with unidirectional communications occurring in horizontal, vertical and diagonal directions. In Figure 5(b) a \( 3 \times 3 \) convolutional layer is assigned to three rows of the PEs. By rearranging this three-row architecture as shown in Figure 5(c), we organise them as a column. When implementing 2D systolic arrays solutions on conventional FPGA column-based chains, it is impossible to use both the input and output dedicated chain connections as they have same source and destination. Figure 5(d) shows a column-based connection which is capable of forwarding the data/result to the next DSP block. This addresses the difficulty of implementing a 2D interconnection on a 1D array, by supporting data forwarding to two DSPs instead of a single one. This is particularly
than computation, this leads improved speed and energy [9], since data movement contributes more to energy consumption. Thus, it is crucial to design efficient hardware where the energy cost is optimized. Our design supports run-time configuration of stream precision. When used to implement convolutional layers, this approach can be applied to both high and low precision operations. To stream low precision inputs, we can pass or flip the AND gate output. Controllable and maskable NAND use both schemes.

Effective for the case where one dimension is small (e.g., 3 elements for $3 \times 3$ convolutional layers).

Current DSP columns are capable of streaming high-precision data over the chains. To stream low precision inputs, we make some minor modifications to the input B register and chaining connections to support both high and low precision data streaming. Also, we modified DSP both input A and B chains to support run-time configurable input data forwarding up to next two DSPs. This is done by bypassing the next DSP to enhance the implementation capabilities for improving data reuse via a small modification to current FPGAs. With our changes, the 18-bit input B can feed both B 27-bit shift registers and their 9-bit LSB portions via both A and B chains. Furthermore, the design supports run-time configuration (Figure 2) of stream precision. When used to implement convolutional layers, these modifications support one high-precision or two low-precision streams for the Stride = 1 and 2 cases.

C. Reuse: Flexible FIFO and Register File

In DNN implementations each input/parameter takes part in many MAC operations, so it is important to cache fetched data. Since data movement contributes more to energy consumption than computation, this leads improved speed and energy [9], [10]. Unfortunately, Xilinx DSP blocks do not support caching of data (this is done using the fine-grained resources or hard memory blocks). Intel DSPs do include a small embedded memory for each 18-bit multiplier, but they cannot be configured at run-time and hence can only be used efficiently for fixed coefficients, making them unsuitable for buffering of data for practical sized DNNs.

We propose a small and flexible first-in-first-out register file (FIFO/RF) to enhance data reuse. This is a wide shift register that can be loaded sequentially and can be read by two standard read ports. The two read port address signals can be provided from outside the DSP block. The first is used inside the DSP and brings the requested and the next data for multiplier and multiplexer units (two 27-bit read ports are needed to feed our multiplier). As RFs are mostly used to buffer a chunk of data inside the DSP, writes always occur as a burst. The other read port is used to select the data for DSP-DSP chaining connections. Using this approach, we arrange the RF as a flexible FIFO. By adjusting the FIFO length, systolic array implementations with different buffering patterns can be implemented. The schematic of our implemented FIFO/RF is given in Figure 2, and operates on input A.

IV. EXPERIMENTAL STUDY

A. Baseline DSP48

As a baseline, we modeled the Xilinx DSP48E2 DSP block using Verilog and synthesized it using SMIC 65-nm technology standard cell by Synopsys Design Compiler 2013.12. Post-synthesis reports show that DSP48E2 timing is consistent with reported speeds for DSP48E1 in Virtex 5 speed grade -1, especially the critical path which is 3.85 and 3.94 ns respectively for our DSP48E2 and Virtex-5 DSP48E1. A comparison with DSP48E1 rather than DSP48E2 was made as the former has generally the same DSP architecture and 65 nm process technology [47]. DSP48E2 is the most recent version including three major architectural upgrades; wider multiplier unit ($27 \times 18$ instead of $25 \times 18$), pre-adder module, and wide XOR circuit. We were not able to compare area since no information is available for the DSP48E1/2 [48].

The baseline DSP48E2 multiplier produces two temporary results, and these are added using the ALU to produce the final MAC output. As a longer critical path is created by the PIR-
DSP partial product summation circuits, we applied parallel computing and carry-lookahead techniques for both multiplier and ALU. It was also necessary to add a new pipeline-register layer to the multiplier unit to reduce the critical path our more complex circuit. Modifications to the ALU also required replacing the DSP48E2 12/24/48-bit SIMD add/sub operations with a 4/8/16/48-bit SIMD which leads to smaller and width-variant ALUs since they must be aligned with the carry propagation blocking points, as shown in Figure 2. We note that the multiplier in the DSP48 is not on the critical path so adding a similar pipeline register does not affect its critical path.

B. Precision (MAC-IP)

Figure 6 and Table II show post synthesis Area, Maximum frequency, and energy per MAC operation results for different configurations of the MAC-IP using the performance optimisation synthesis strategy.

Table III Configuration #0 shows our synthesised DSP48E2 area and maximum frequency. Configurations #1 to #3 results obtained by simply replacing the multiplier and ALU units in the DSP48E2 with the MAC=IP. Upgrading the multiplier to a 27×18C32D2 MAC-IP, leads to improvements in MAC capabilities of ×6, ×12, ×24 times for 9, 4, 2-bit MAC operations respectively, at the cost of a 14% increase in area.

C. Interconnect and Reuse (PIR-DSP)

Table III Configuration #4 is produced by adding the interconnect optimisation to Configuration #3. Configuration #5 is the final implementation of PIR-DSP which adds the reuse optimisation. As in the DSP48 data sheet, the reported $F_{max}$ to the P output omits the wide XOR and pattern detector circuits of Figure 2.

To evaluate the effectiveness of our proposed data movement modifications for low-precision computations, we focused on the total run-time energy required by implementing low-precision versions of some well-cited embedded CNNs.

We extracted the read and write energy using Xilinx Power Estimator (XPE) for BRAM and LUT blocks on the Virtex-5 FPGA. $E_{\text{BRAM, read}}$ and $E_{\text{BRAM, write}}$ per byte were estimated for an 18-bit wide memory configuration (most efficient way to use BRAMs). To estimate the energy associated with moving data from an off-DSP register file (RF) and shift-register (SR), we configured the LUTs respectively as RAM with Fanout = 4 (for broadcasting), and shift register with Fanout = 1 (streaming) (Table IV). Using results for small register files in [49], [50], and [51], we estimated our embedded 4×2 30-bit RF read & write energy to be 1.1 pJ/byte. RF width and size are selected respectively, to fully feed the multiplier/pre-adder in high/low-precision and to be similar to Intel DSP block read-only RFs which are configured in two 8×8-bit memories per DSP. To estimate input B energy which operates as a SR and a normal register we used results for high-performance [52] and low energy flip-flops [53] (FF) to obtain estimates of 180 fJ and 90 fJ respectively. Energy required to transfer data from DSP-DSP was obtained from [54], and scaled to 65nm technology, to obtain 2 pJ per byte. Using the energy ratios from Table II, energy consumption for 9/4/2-bit MAC operations are 89/44/22 × that of a 9-bit register. Table V summarises the estimated energy ratios for data movement. We further assume that all elements (except the MAC) scale linearly with wordlength.

D. Implementation of Convolutions

We now describe implementations of standard and DW convolutional layers, using a 3×3 DW convolution layer as a case study. According to Equation 2, output channels can be computed in parallel. We assumed input and weight parameters are located in BRAMs and results will be written back to

### Table II

<table>
<thead>
<tr>
<th>MAC Model</th>
<th>Area ratio</th>
<th>Fmax (MHz)</th>
<th># of MAC / Energy per MAC (pJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>27×18-MAC</td>
<td>1</td>
<td>763</td>
<td>1/28.4/1.00 1/28.4/1.00 1/28.4/1.00</td>
</tr>
<tr>
<td>27×18C32D0</td>
<td>1.46</td>
<td>730</td>
<td>5/76.3/6.63 6/6.3/6.63 6/6.3/6.63</td>
</tr>
<tr>
<td>27×18C32D1</td>
<td>1.86</td>
<td>671</td>
<td>1/43.9/6.73 12/3.7/12.37 12/3.7/12.37</td>
</tr>
<tr>
<td>27×18C32D2</td>
<td>1.70</td>
<td>538</td>
<td>1/47.9/6.80 12/4/24.0 24/2.0 24/2.0</td>
</tr>
<tr>
<td>27×27C33D0</td>
<td>2.12</td>
<td>714</td>
<td>1/54.1/9.60 9/6.0/9.60 9/6.0/9.60</td>
</tr>
<tr>
<td>27×27C33D1</td>
<td>2.21</td>
<td>581</td>
<td>1/59.5/9.66 18/3.3/18.33</td>
</tr>
<tr>
<td>27×27C33D2</td>
<td>2.36</td>
<td>380</td>
<td>1/90.8/18.50 18/5.0/36.25</td>
</tr>
</tbody>
</table>

### Table III

<table>
<thead>
<tr>
<th># of DSP Version</th>
<th>Area Post Synth. ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 DSP48E2</td>
<td>25419</td>
</tr>
<tr>
<td>1 + M27×18C32D0 MAC-IP</td>
<td>28638</td>
</tr>
<tr>
<td>2 + M27×18C32D1 MAC-IP</td>
<td>28838</td>
</tr>
<tr>
<td>3 + M27×18C32D2 MAC-IP + interconnect</td>
<td>29097</td>
</tr>
<tr>
<td>4 + M27×18C32D2 MAC-IP + interconnect</td>
<td>29972</td>
</tr>
<tr>
<td>5 PIR-DSP=MAC-IP+ + interconnect + reuse</td>
<td>32505</td>
</tr>
</tbody>
</table>

**Figure 6.** PIR-DSP synthesis results for different MAC-IP configurations. The PIR-DSP case includes all our optimisations.
TABLE IV
Estimation of BRAM, Off-DSP RF and RS Read/Write access energy 9-bit word on a Xilinx XC5VLX155T extracted from XPE tool (1fJ).

<table>
<thead>
<tr>
<th>BRAM Metrics</th>
<th>Method</th>
<th>BRAM Output width</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_{\text{read}}$</td>
<td>$100%$ Read usage</td>
<td>18 9 4 1</td>
</tr>
<tr>
<td>$E_{\text{write}}$</td>
<td>$100%$ Write usage</td>
<td>8.45 15.8 32.3 116</td>
</tr>
<tr>
<td>$E_{\text{RAM}}$ (E_B)</td>
<td>$E_{\text{read}}$+$E_{\text{write}}$</td>
<td>18.43 33.7 67.9 244</td>
</tr>
<tr>
<td>LUT FANOUT</td>
<td>Method</td>
<td>LUT FanOut</td>
</tr>
<tr>
<td>$E_{\text{RF}}$ (Off-DSP)</td>
<td>LUT as RAM</td>
<td>4 3 2 1</td>
</tr>
<tr>
<td>$E_{\text{SR}}$ (Off-DSP)</td>
<td>LUT as $SR$</td>
<td>4.92 4.59 4.27 3.95</td>
</tr>
</tbody>
</table>

TABLE V
Data movement energy ratios in 65 nm technology ($1 \times = 90fJ$).

<table>
<thead>
<tr>
<th>Energy Ratio</th>
<th>FF</th>
<th>SRs</th>
<th>RFs</th>
<th>Chain 2</th>
<th>RF</th>
<th>SR</th>
<th>BRAM(B)</th>
<th>MAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF SRs RFs Chain 2 RF SR BRAM(B) MAC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

BRAMs. In an implementation on conventional DSPs [55], weight stationary data flow was used, with each input feature map element fetched once from BRAMs and then streamed over off-DSP SRs. Weight parameters are fetched once from BRAMs and saved in DSP registers. Each filter and input element are respectively used $F_h \times F_w$ and $K_h \times K_w$ times. The average energy for the described data flow where $E_{\text{MAC}}$ is the energy consumption of the MAC computation is

$$E_{\text{conv}} = E_{\text{Input}} + E_{\text{Weight}} + E_{\text{MAC}}$$

$$= \left( \frac{E_B}{K_w K_h} + E_{SR} + E_{FF} \right) + \left( \frac{E_B}{F_w} + E_{FF} \right) + E_{\text{MAC}}$$

(6)

1) Depth-wise Convolution: For a PIR-DSP implementation, inspired by the Eyeriss architecture [9], we mapped computation of multiple rows of output channels to a three-cascaded PIR-DSP (Figure 7). Each PIR-DSP can compute 2/4/8 sets of three-MAC operations for 9/4/2-bit precision. Each three-MAC operation can be used for a row of a $3 \times 3$ DW kernel. Cascading three PIR-DSPs, we can sum the partial outputs to produce the final output feature map elements. As illustrated in Figure 7 for 9-bit precision, each PIR-DSP receives two streams of 9-bit data (as each PIR-DSP can compute two parallel three-MAC operations). The three-cascaded PIR-DSPs can forward two of their streams to the next three-cascaded PIR-DSP over the DSP-DSP chains, and we can implement $K$ rows of 2/4/8 channels of the output matrix for 9/4/2-bit precision using a column of 3K PIR-DSPs. For this case, $E_{\text{Input}}$ becomes

$$E_{\text{Stream,Input}} = \frac{E_B + (N_{\text{OF}})E_{\text{Chain}}}{K_h K_w} + E_{SRs}$$

(7)

where NoF is the number of forwarding over chains for each input stream (2 in our case as each row of the input stream is involved in three rows of output feature map). To implement other kernel sizes, we use a kernel tiling approach with tile size of $3 \times 3$, $2 \times 3$, and $1 \times 3$ which are respectively the computation capabilities of a three-cascaded, two-cascaded, and a PIR-DSP. Thus a $5 \times 5$ kernel can be implemented using $2 \times$ three-cascaded DSPs and $2 \times$ two-cascaded DSP groups where NoF is 6.

2) Standard Convolution: For the case of standard convolution, our RF reuse reduces $E_{\text{Input}}$ by a factor of $R_{\text{size}}$ (last line of Table VI). The calculated access energy ratio in the last column indicates that PIR-DSP uses $31\%$ of the data access energy for a middle bottleneck layer of MobileNetv2 [12] which applies 192 depth-wise $3 \times 3$ filters on an input feature map of shape $56^2 \times 192$.

3) Point-wise Convolution: For a PW convolution, each input channel can be streamed into a DSP to be multiplied by corresponding weight parameter, producing a partial result which is cascaded and summed to produce an entry of the output feature map. In a PIR-DSP implementation, we assign three channels of input and three corresponding channels of 2/4/8 PW kernels to a PIR-DSP, depending on precision. PIR-DSP using 2, 4, or 8 three-MAC operations computes partial results of each filter on the same input stream in parallel (the stream includes an element of three channels of input feature map in each cycle). By cascading we can compute 2, 4, or
TABLE VI
READ ACCESS ENERGY FOR STANDARD/DW/PW CONV. LAYER PER MAC (BASELINE IMPLEMENTATION USES OFF-DSP RESOURCES TO STREAM INPUT OVER SAVED WEIGHTS IN DSP REGISTERS).

<table>
<thead>
<tr>
<th>Method</th>
<th>EInput</th>
<th>EOutput</th>
<th>Ratio%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>EInput + E Skip + E Skip / K in</td>
<td>EOutput + E Skip / K out</td>
<td>100</td>
</tr>
<tr>
<td>Stream</td>
<td>EInput + E Skip + E Skip / K in</td>
<td>EOutput + E Skip / K out</td>
<td>45</td>
</tr>
<tr>
<td>Str.&amp;RF</td>
<td>EInput + E Skip + E Skip / K in</td>
<td>EOutput + E Skip / K out</td>
<td>31</td>
</tr>
</tbody>
</table>

TABLE VII
ENERGY RATIO OF PIR-DSP OPTIMISATIONS FOR 9/4/2-BIT PRECISION (PERCENT)(BASELINE = 100/100/100).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P R I</td>
<td>baseline</td>
<td>baseline</td>
<td>baseline</td>
<td>baseline</td>
</tr>
<tr>
<td>✗ ✗ ✓</td>
<td>39/26/20</td>
<td>38/26/20</td>
<td>40/28/22</td>
<td></td>
</tr>
<tr>
<td>✓ × ✓</td>
<td>33/20/14</td>
<td>33/21/15</td>
<td>31/20/14</td>
<td></td>
</tr>
<tr>
<td>✓ ✗ ✓</td>
<td>31/19/13</td>
<td>31/19/13</td>
<td>29/17/12</td>
<td></td>
</tr>
</tbody>
</table>

8 six-MAC operations (computing six elements of the PW kernels). Also, as illustrated in the right hand part of Figure 7 for 9-bit precision, each two-cascaded PIR-DSP can forward their streams to next two-cascaded DSP which leads to energy reduction as summarised in Table VI. Thus, PIR-DSP uses saved weights and performs a MAC with the 2/4/8 3-channel weight parameters which are saved in 27-bit registers. Furthermore, the RF improves input data reuse. By applying the equations to a middle bottleneck layer of MobileNet-v2 (which includes 192 PW 1 × 1 × 32 filters on 562 × 32 input feature map), our proposed optimisations can reduce the read access energy to 44% of the original value.

A similar analysis was applied to all layers of some common embedded DNN models, the results in Table VII are obtained. For example, when applying all our optimisations to MobileNet-v2 [12], energy is reduced to 31/19/13% of the original value for 9/4/2-bit precision.

E. Comparison with Previous Work

BitFusion [56] is an ASIC DNN accelerator, supporting multi-precision MACs. The reported area is for a computation unit in 45-nm technology, comprising 16 BitBricks, each of which is a 2-bit plus sign multiplier. This is similar to our 27 ×18C32D2 MAC-IP (Table II), although BitFusion is more flexible as it supports more variations including 2 ×4, 2 ×8 and 4 ×8. Table VIII compares Performance per Area (PPA). We used the maximum frequency reported for a same implementation, DSP48E1, in three FPGAs, Virtex5/6/7, normalized to feature size [57] (area is scaled by 1/0.66/0.3 and maximum frequency by 1/1.1/1.35 respectively for 65/45/28 nm). BitFusion only applies the chopping technique, leading to high area overhead. The introduction of recursive decomposition better supports low and high-precision MAC operations.

Boutros et. al. proposed improvements to the Intel DSP block [44], and is capable of 27 ×27 and reduced precision MACs down to 4-bit. In comparison, PIR-DSP can support precisions down to 2 bits, has better performance at 8 × 8 bits and lower, is generated using a flexible module generator, but is worse at 16 × 16 and higher PPA. It is not possible to compare energy but we would expect Boutros to be similar to the Baseline case in Table VI with PIR-DSP having significant advantages due to the interconnect and reuse optimisations.

V. CONCLUSION

We proposed PIR-DSP which incorporates precision, interconnect and reuse optimisations to better support 2-dimensional low-precision DNN applications. When applied to the implementation of embedded DNNs, for which the bottleneck is the standard, PW and DW convolutions, it was shown that our DSP block architecture can significantly reduce the energy consumption of low-precision implementations, albeit requiring an extra cycle of latency and a 28% area overhead.

Future work will include optimising the critical path of the PIR-DSP by providing a bypass path for the unused pre-adder. This could enhance frequency by 25% or be used to remove the extra cycle of latency introduced by our additional pipeline stage.


[37] Virtex-5 FPGA Data Sheet: DC and Switching Characteristics, Xilinx, 6 2016, v5.5


