Low Complexity Multiply-Accumulate Units for Convolutional Neural Networks with Weight-Sharing

JAMES GARLAND and DAVID GREGG, Trinity College Dublin and Trinity College Dublin, Ireland

Convolutional neural networks (CNNs) are one of the most successful machine-learning techniques for image, voice, and video processing. CNNs require large amounts of processing capacity and memory bandwidth. Hardware accelerators have been proposed for CNNs that typically contain large numbers of multiply-accumulate (MAC) units, the multipliers of which are large in integrated circuit (IC) gate count and power consumption. "Weight-sharing" accelerators have been proposed where the full range of weight values in a trained CNN are compressed and put into bins, and the bin index is used to access the weight-shared value. We reduce power and area of the CNN by implementing parallel accumulate shared MAC (PASM) in a weight-shared CNN. PASM re-architects the MAC to instead count the frequency of each weight and place it in a bin. The accumulated value is computed in a subsequent multiply phase, significantly reducing gate count and power consumption of the CNN. In this article, we implement PASM in a weight-shared CNN convolution hardware accelerator and analyze its effectiveness. Experiments show that for a clock speed 1GHz implemented on a 45nm ASIC process our approach results in fewer gates, smaller logic, and reduced power with only a slight increase in latency. We also show that the same weight-shared-with-PASM CNN accelerator can be implemented in resource-constrained FPGAs, where the FPGA has limited numbers of digital signal processor (DSP) units to accelerate the MAC operations.

CCS Concepts: • **Hardware → Arithmetic and datapath circuits**; **Hardware accelerators**; **Chip-level power issues**; **Datapath optimization**;

Additional Key Words and Phrases: CNN, power efficiency, multiply accumulate, arithmetic hardware circuits, ASIC, FPGA

ACM Reference format:
James Garland and David Gregg. 2018. Low Complexity Multiply-Accumulate Units for Convolutional Neural Networks with Weight-Sharing. *ACM Trans. Archit. Code Optim.* 15, 3, Article 31 (August 2018), 24 pages.
https://doi.org/10.1145/3233300

This research is supported by Science Foundation Ireland, Project 12/IA/1381. We also thank the Institute of Technology Carlow, Carlow, Ireland, for their support.

This article is an expanded version of a short, four page article that appeared in IEEE Computer Architecture Letters (CAL) ([8]). IEEE CAL's publication policy is that "due to the short format, we expect that publication in IEEE CAL should not preclude subsequent publication in top-quality conferences or full-length journals." Our earlier IEEE CAL short article proposed our parallel accumulate shared MAC (PASM) unit for multiply-accumulate operations and provided an evaluation of the unit in isolation using an application-specific integrated circuit (ASIC) design flow. In contrast, the current article provides much greater detail and analysis and evaluates our PASM unit in the context of a Convolutional neural network (CNN) hardware accelerator rather than in isolation. In the current article, we also study the effectiveness of PASM for a CNN accelerator on field programmable gate arrays (FPGAs) and provide experimental results.

Authors' addresses: J. Garland and D. Gregg, School of Computer Science and Statistics, Trinity College Dublin, Westland Row, Dublin D02 DP70, Ireland; emails: jgarland@tcd.ie, david.gregg@cs.tcd.ie.

1544-3566/2018/08-ART31 $15.00
https://doi.org/10.1145/3233300

1 INTRODUCTION

Convolutional neural networks (CNNs) are used on a daily basis for image [14], speech [13], and text recognition [16], and their use and application to different tasks is increasing at a very rapid rate. However, CNNs require huge memory storage and bandwidth for weight data and large amounts of computation that would push to extremes the battery, computation, and memory in mobile embedded systems. Researchers [10, 11] have proposed methods of quantizing and dictionary compressing the weight data to reduce the memory bottleneck and bus bandwidth. Others [21, 22] have proposed various different CNN hardware accelerators implemented in both field programmable gate arrays (FPGAs) and application-specific integrated circuit (ASICs) that may contain hundreds to thousands of parallel hardware multiply-accumulate (MAC) units to increase the computational performance. This increase in computational performance comes at the great expense of power as the MAC units contain a multiplier, each of which consumes large numbers of logic gates and high-power consumption in an ASIC [18].

CNNs are extensively used in an inference mode [14, 20] to infer that, for example, a dog can be found within an image. However, the CNN must first be trained. Training the CNN involves incrementally modifying the "weight" values associated with connections in the neural network and retraining until a satisfactory error rate has been achieved [15]. At this point, the network is considered trained, meaning that no further updates of weight values are required, and the trained network can be deployed for inference to the field. In their research, Han et al. [10, 11] found that in a fully trained CNN, similar weight values occur many times. They proposed scalar quantization of the weight data by clustering around centroids to dictionary compress the weights into bins. They found that between tens to hundreds of weight values were sufficient in network inference while maintaining the high accuracy rate. They encode the compressed weights with an index that specifies which of the shared weights should be used. This dictionary compression of the weight data reduces the required size and memory bandwidth required for the network. They demonstrate that their weight-shared values can be stored on-chip consuming 5pJ per access rather than in off-chip dynamic RAM (DRAM) that consumed 640pJ per access when implemented on a central processing unit (CPU)/graphics processor unit (GPU) system. Weight sharing does not reduce the number of MAC operations required; it reduces only the weight data storage and bandwidth requirement.

Building on Han et al's. [10, 11] research, we propose a re-architected MAC circuit of the weight-shared CNN aimed at hardware accelerators. Rather than computing the sum-of-products (SOP) in the MAC directly, we instead count how many times each of the weight indexes appears and store the corresponding image value in a register bin, thus replacing the hardware multipliers with counting, selection, and accumulation logic. After this weighted histogram accumulation phase, a post pass multiplication is performed of the accumulated image values in bins with the corresponding weight value of that bin. We call this accelerator optimization the parallel accumulate shared MAC (PASM). To evaluate PASM performance we implement PASM in a convolution layer of a weight-shared CNN accelerator. Where weight bin numbers are small and channel numbers are large, the counting and selection logic can be significantly smaller and lower power than the corresponding multiply-accumulate circuit. We also show that PASM is beneficial when implemented in a resource-constrained FPGA as PASM consumes fewer block RAMs (BRAMs) and DSP units for the MAC operations in the FPGA.

The rest of this article is organized as follows. Section 2 gives some background on CNN accelerators and introduces the PASM and how it compares to other CNN accelerators. Section 3 shows how our PASM is implemented in a convolution layer accelerator with examples compared to a weight-shared accelerator. Section 4 describes how a weight-shared-with-PASM convolution

```
image[C][IH][IW], weight[M][C][KY][KX];
outFeat[M][OH][OW];

for (ihIdx=(KY/2); ihIdx<(IH-(KY/2)); ihIdx+=Stride) {
  for (iwIdx=(KX/2); iwIdx<(IW-(KX/2)); iwIdx+=Stride) {
    for (mIdx=0; mIdx<M; mIdx++) {
      summands = 0;
      for (cIdx=0; cIdx<C; cIdx++) {
        for (kyIdx=0; kyIdx<KY; kyIdx++) {
          for (kxIdx=0; kxIdx<KX; kxIdx++) {
            imVal = image[cIdx][((ihIdx+kyIdx)-(KY/2))][((iwIdx+kxIdx)-(KX/2))];
            kernVal = kernel[mIdx][cIdx][kyIdx][kxIdx];
            summands += imVal * kernVal;
          }
        }
      }
      outFeat[mIdx][ihIdx/Stride][iwIdx/Stride] = summands;
    }
  }
}
```

Fig. 1. Simplified pseudo-code of a convolution layer.

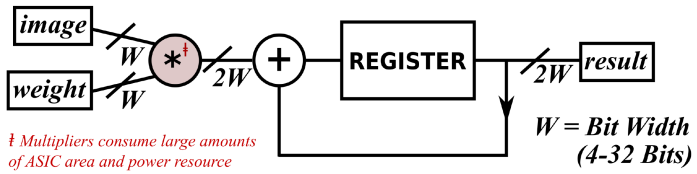


Fig. 2. Simple MAC block diagram.

accelerator is designed and implemented in an ASIC at 45nm clocked at 1GHz and in a Zynq FPGA clocked at 200MHz. Section 5 presents the experimental results showing latency, power, and area projections for both FPGA and ASIC. Section 6 reviews related work and Section 7 draws conclusions.

2 DNN CONVOLUTION WITH DICTIONARY-ENCODED WEIGHTS

2.1 CNN Accelerators

A deep neural network (DNN) contains convolution layers, activation function layers (such as a sigmoid or rectified linear unit (ReLU)), and pooling layers. Up to 90% of the computation time of a CNN is taken up by the convolution layers [4]. Within the convolution layer, there are many thousands of MAC operations, as shown in the pseudo code in Figure 1. The convolution operator has an input image of dimensions $IH \times IW$ and C channels and is convolved with M kernels (typically 3 to 832 [21]) of dimension $KY \times KX$ and C channels at a stride of S to create an output feature map of $OH \times OW$ and M channels. The loops can be unrolled into parallel MAC units and implemented in hardware [22] to accelerate the convolution.

A MAC unit (see Figure 2) is a sequential circuit that accepts a pair of numeric values (image and weight values) of a predefined bit width and type (e.g., 32-bit fixed point integers), computes their product and accumulates the result in the local accumulator register each clock cycle. The locality of the accumulator register reduces routing complexity and clock delays within the MAC.

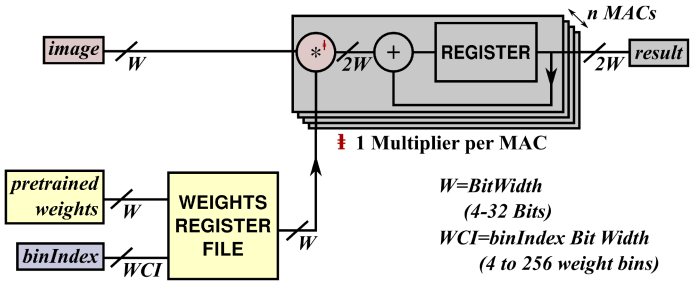


Fig. 3. Simplified weight-shared MAC block diagram.

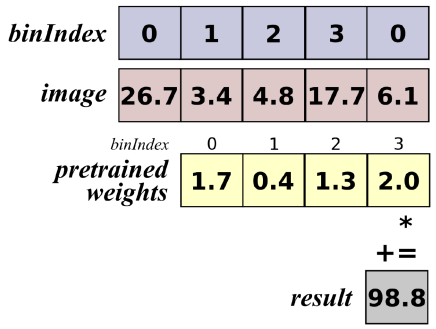


Fig. 4. Simplified weight-shared MAC example.

Han et al. [10, 11] propose a weight-sharing architecture to reduce the power and memory bandwidth consumption of CNNs. They found that similar weight values occur multiple times in a trained CNN. By binning the weights and retraining the network with the binned values, they found that just 16 weights were sufficient in many cases. They encode the weights by replacing the original numeric values with a 4-bit index that specifies which of the 16 shared weights should be used. This greatly reduces the size of the weight matrices. Figure 3 shows simplified weight-sharing decode logic coupled with multiple MAC units of the CNN. When the kernel input is encoded using weight sharing, an extra level of indirection is required to index and access the actual weight value from the weights register file.

Figure 4 shows an example of the weight-shared MAC in operation. Each ***image*** value is streamed in, and its corresponding ***binIndex*** is used to access the pretrained weight against which to multiply and accumulate into the result register. Figure 4 shows how ***image*** value 26.7 is multiplied-accumulated with the pretrained weight 1.7 indexed by ***binIndex*** 0. Next 3.4 is multiplied-accumulated with the pretrained weight 0.4 indexed by ***binIndex*** 1. This continues until finally multiplying-accumulating image value 6.1 with pretrained weight value 1.7 of bin 0 to give:

$$\textbf{\textit{result}} = (26.7 \times 1.7) + (3.4 \times 0.4) + (4.8 \times 1.3) + (17.7 \times 2.0) + (6.1 \times 1.7) = 98.8.$$

In both a simple MAC (Figure 2) and a weight-shared MAC (Figure 3) the multiplier is the most expensive unit in terms of floor area (i.e., large numbers of gates) and power consumption in an ASIC or numbers of DSP units in an FPGA. As a large number of MAC units are used in a parallel weight-shared CNN hardware accelerator, the overall area and power is likely to be large.

Weight sharing is an important factor in implementing CNN accelerators in an off-line embedded, low-power device. Han et al. [10, 11] show that when pruning, quantization, weight-sharing, and Huffman coding are all used together in an AlexNet [14] CNN accelerator, the weight data

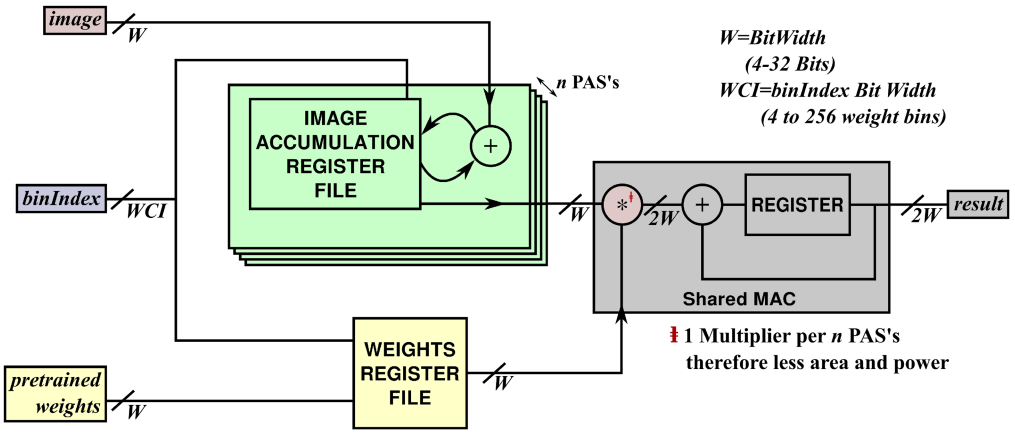


Fig. 5. PASM showing PAS unit followed by a shared MAC.

required is reduced from 240MB to 6.9MB, a compression factor of 35×. Unfortunately, they do not provide results for the effect of weight sharing alone, without these other optimizations. When they apply similar pruning, quantization, weight-sharing, and Huffman to the VGG-16 [20] CNN accelerator, the weight data are reduced from 552MB to 11.3MB, a 49× compression ratio. The fully connected layers dominate the model size by 90%, but Han et al. [10, 11] show that these layers compress by up to 96% of weights pruned in VGG-16 CNN. These newly weight-shared CNNs run 3× to 4× faster on a mobile GPU while using 3× to 7× less energy with no loss in classification accuracy. As the number of free parameters being learned is reduced in a weight-shared CNN, the learning efficiency is greatly increased and allows for better generalization of CNNs for vision classification.

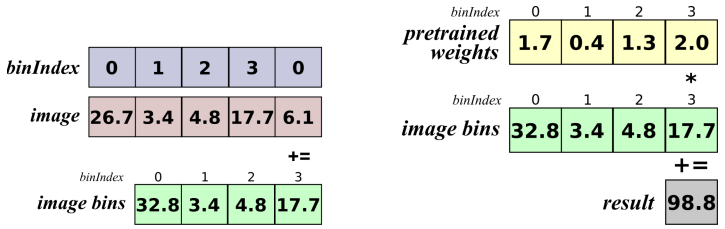
The trend is towards increasingly large networks, increasing the number of layers such as ResNet [12] or increasing the convolution types within each layer such as GoogLeNet [21]. Weight sharing is one method that is getting increased research focus to reduce the overall weight data storage and transfer so that the networks can be implemented on off-line mobile devices.

CNN hardware accelerators typically use 8-, 16-, 24- or 32-bit fixed point arithmetic [2]. A combinatorial W-bit multiplier requires $O(W^2)$ logic gates to implement that makes up a large part of the MAC unit. Note that sub-quadratic multipliers are possible but are inefficient for practical values of W [6].

2.2 The PASM Concept

We propose to reduce the area and power consumption of MAC units by re-architecting the MAC to do the accumulation first, followed by a shared post-pass multiplication. Our new PASM accelerator is shown in Figure 5. Rather than computing the SOP in the MAC directly, PASM instead counts how many times each B-bin weight-shared index appears and accumulates the corresponding W-bit ***image*** value in the corresponding B weight-shared bin register indexed by the ***binIndex***. PASM has two phases: (1) accumulate the image values into the weight bins (known as the parallel accumulate and store (PAS)) and (2) multiply the binned values with the weights (completing the PASM).

Figure 6(a) shows an example of the accumulation phase. Our PAS unit is a sequential circuit that consumes a pair of inputs each cycle. One input is an ***image*** value, and the other is the ***binIndex*** of the weight value in the dictionary of weight encodings. The PAS unit has a set of B accumulators, one for each entry in the dictionary of weight encodings. The accumulators are initially set to



(a) Phase 1: As each image value is streamed in, its associated bin index is also streamed so that the image values can be accumulated into correct bins.

(b) Phase 2: Each bin accumulated value is multiplied with its corresponding pre-trained weight value to produce the final result.

Fig. 6. PASM in operation.

zero. Each time the PAS consumes an input pair, it adds the ***image*** value to the accumulator with index ***binIndex***. For example, when the leftmost pair of inputs in Figure 6(a) are consumed, the ***image*** value 26.7 is added onto accumulator numbered ***binIndex*** = 0. Next 3.4 is accumulated into bin 1. This continues until finally accumulating 6.1 into bin 0 to give 26.7 + 6.1 = 32.8. This accumulated result tells us that the weight stored in dictionary location 0 has been paired with an accumulated ***image*** value of 32.8. For the accumulation phase, the actual weight value stored in dictionary location 0 does not matter. We are simply computing a weighted histogram of the dictionary weight indices.

In the second phase, the histogram of weight indices is combined with the actual weight values to compute the result of the sequence of multiply-accumulate operations. Figure 6(b) demonstrates the multiply phase, multiplying-accumulating bin 0 ***pretrained weight*** with bin 0 accumulated ***image*** value, giving $32.8 \times 1.7 = 55.76$. The contents of ***pretrained weight*** bin 1 is multiplied-accumulated with ***image*** bin 1 value and so on until all the corresponding bins are multiplied-accumulated into the ***result*** register, giving 98.8, the same result found by the weight shared MAC, Figure 4.

This second multiply stage can be implemented using a traditional MAC unit that is shared between several PAS units. Several MAC units can be replaced by the same number of PAS units sharing a single MAC. For example, consider the case where we must compute many multiply-accumulate sequences, where each sequence consumes 1,024 pairs (image and weight) of values. A fully-pipelined MAC unit is a sequential circuit that will typically require a little over 1,024 cycles to compute the result.

If we have four such MAC units, then we can compute four such results in parallel, again in around 1,024 cycles. If the weight data has been quantized and dictionary encoded to just, say, 16 values, then we could use PAS units with $B = 16$-bins to perform the accumulate phase of the PASM computation. Four such fully pipelined PAS units could perform the accumulation phase in around 1,024 cycles. However, the accumulation phase of the PASM does not give us the complete answer. We also need to perform the multiply phase, which involves multiplying and accumulating $B = 16$ values in this example. If each PAS unit had its own MAC unit, then the multiply phase would take around 16 cycles for a total of 1,024 + 16 = 1,040 cycles for the entire multiply accumulate operation. However, in this example, the four parallel PAS units share a single MAC unit with the result that the total time will be $1{,}024 + 4 \times 16 = 1{,}088$ cycles. PASM can have higher throughput when compared to the standard MAC due to the PAS units being much smaller than the MAC for small values of B, up to about $B = 16$.

Table 1. Complexity of MAC, Weight-shared MAC and PAS

Sub Component	Gates	Simple MAC	Weight Shared MAC	PAS
Adder	$O(W)$	1	1	1
Multiplier	$O(W^2)$	1	1	
Weight Register	$O(W)$	0	B	
Accumulation Register	$O(W)$	1	1	B
File Port	$O(WB)$		1	2

2.3 PASM Accelerator

Table 1 compares the gate counts of the sub components of a simple MAC, a weight-shared MAC and a PAS. The *gates* column shows the circuit complexity in gates of each sub-component, assuming fixed-point arithmetic. The bit-width of the data is W and the number of bins is B in the weight-shared designs. For example, a simple MAC unit contains an adder ($O(W)$ gates), a multiplier ($O(W)^2$) gates) and a register ($O(W)$ gates). A weight shared MAC also needs a small register file with B entries to allow fast mapping of encoded weight indices to shared weights. The PAS needs a read and write port due to the interim storage of the accumulation results that need to be read by the post pass multiplier, whereas the MAC only needs a write port.

From Table 1, we can also see that the efficiency of PAS depends on a weight-sharing scheme where the number of bins, B, is much less than the total number of possible values that can be represented by a weight value, that is 2^W. For example, if we consider the case of $W = 16$, then in the absence of weight sharing, a PAS would need to deal with the possibility of 2^{16} different weight values, requiring 2^{16} separate bins. The hardware area of these bins is likely to be prohibitive. Therefore, PAS is effective where the number of bins is much lower than 2^W.

2.4 Evaluation of PASM as a Stand-alone Unit

We design an accelerator unit to perform a simplified version of the accumulations in Figure 5. Our accelerator accepts 4 image inputs and 4 shared-weight inputs each cycle and uses them to compute 16 separate MAC operations each cycle. The weight-shared version performs these operations on 16 weight-shared MAC units (16-MAC). Our proposed PASM unit has 16 PAS units and uses 4 MAC units for post-pass multiplication (16-PAS-4-MAC). Both the weight-shared and weight-shared-with-PASM accelerators are coded in Verilog 2001 and synthesized to a flat netlist at 100MHz with a short 0.1ns clock transition time targeted at a 45nm process ASIC. We measure and compare the timing, power, and gate count in both designs for the same corresponding bit widths and same numbers of weight bins.

The standard 16-MAC and the proposed 16-PAS-4-MAC each have W-bit ***image*** and ***weight*** inputs and the 16-PAS-4-MAC has a WCI-bit ***binIndex*** input to index into the $B = 2^{wci}$ weight bins. The designs are coded using integer/fixed point precision numbers. Both versions are synthesized to produced a gate level netlist and timing constraints designed using Synopsys design constraint (SDC) [7] so that both designs meet timing at 100MHz.

Cadence Genus (version 15.20 - 15.20-p004_1) is used for synthesizing the register transfer logic (RTL) into the OSU FreePDK 45nm process ASIC and applying the constraints to meet timing. Genus supplies commands for reporting approximate timing, gate count, and power consumption of the designs at the post-synthesis stage. The "report timing," "report gates," and "report power" commands of Cadence Genus are used to obtain the results for both 16-MAC and 16-PAS-4-MAC accelerators. Graphs of the gate count and power consumption results are produced for the two

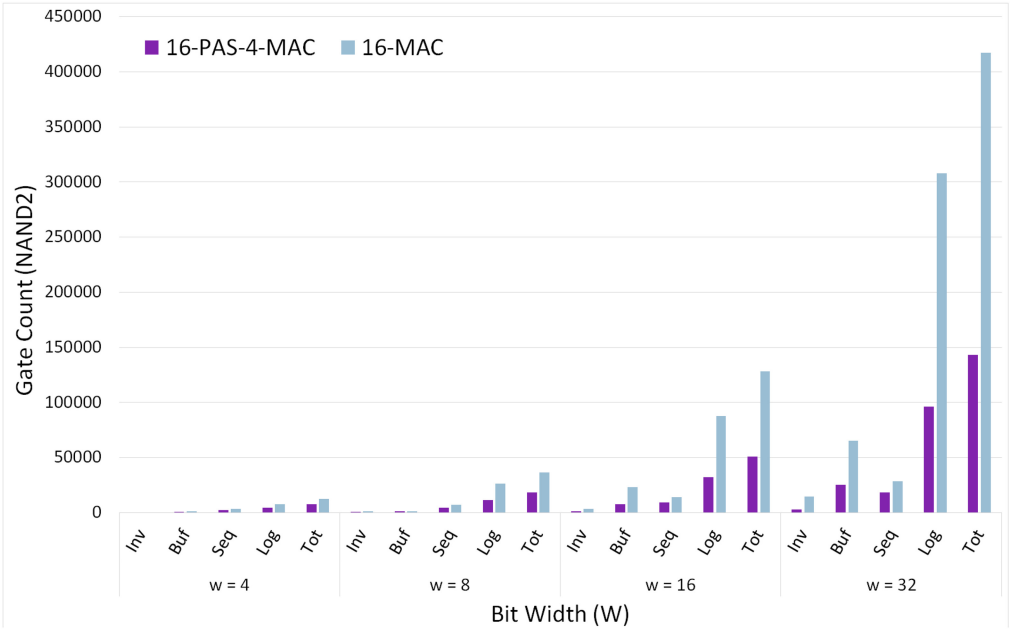


Fig. 7. Logic gate count comparisons (in NAND2X1 gates) for W = 4-, 8-, 16-, 32-bits wide 16-MAC and 16-PAS-4-MAC for B = 16 weight bins; lower is better.

different designs at different bit widths and different numbers of weight bins, showing that the PASM is consistently smaller and more efficient than the weight-sharing MAC.

Figure 7 shows comparisons of the logic resource requirements of a B = 16 shared-weight-bin 16-PAS-4-MAC and 16-MAC for varying W-bit widths. Gate counts are normalized to a NAND2X1 gate. The PASM uses significantly fewer logic gates. For example, for W = 32-bit wide the 16-PAS-4-MAC is 35% smaller in sequential logic, 78% smaller in inverters, 61% smaller in buffers and 68% smaller in logic, an overall 66% saving in total logic gates. The PASM requires more accumulators for the B-entry register file, but otherwise overall resource requirements are significantly lower than that of the MAC.

Figure 8 shows comparisons of power consumption of the accelerators. 16-PAS-4-MAC's power is lower than the weight-shared 16-MACs and the gap grows with increasing W-bit width. For example, for the W = 32-bit versions of each design, the 16-PAS-4-MAC consumes 60% less leakage power, 70% less dynamic power and 70% less total power than that of the 16-MAC version.

Figure 9 shows the effect of varying the number of bins from B = 4 to B = 256, with gate counts normalized to a NAND2X1. For bit width W = 32 and B = 16-bins the 16-PAS-4-MAC utilization has 35% fewer sequential gates, 78% fewer inverters, 62% fewer buffers and 69% fewer logic and 66% less total logic gates compared to the 16-MAC design. However, at B = 256, PASM registers and buffers are less efficient than the MAC.

The 16-PAS-4-MAC also consumes 61% less leakage power, 70% less dynamic power, and 70% less total power (Figure 10). More details can be found in our original article, [8].

3 PASM IN A CNN ACCELERATOR

In this article, we asked the question would PASM offer similar power and area savings when implemented in a layer of a CNN accelerator and how would it affect performance of the convolution

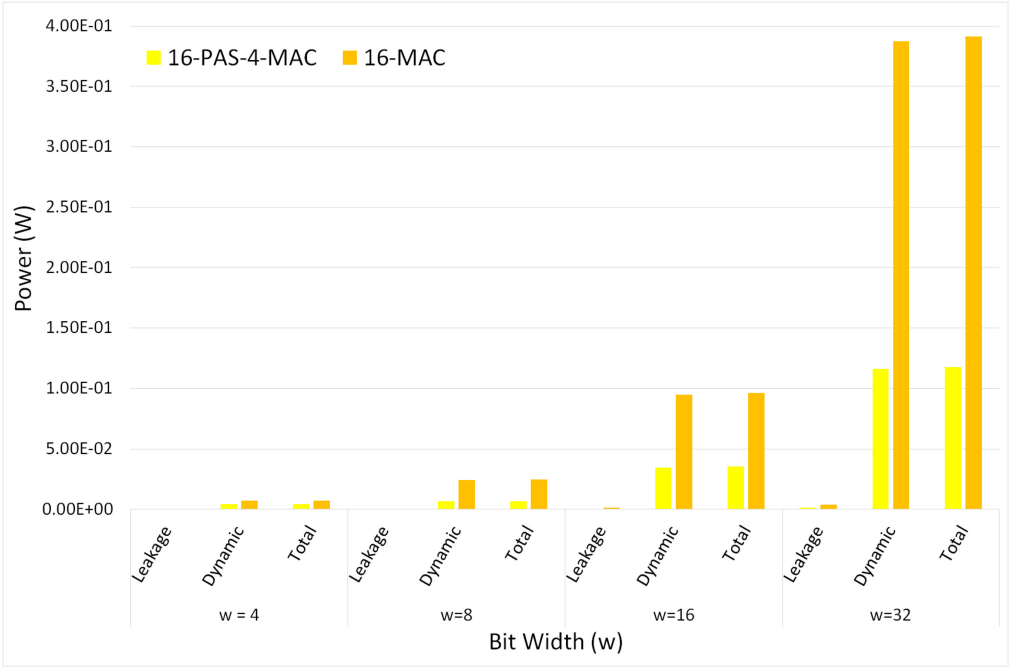


Fig. 8. Power consumption (in W) comparisons for $W = 4, 8, 16, 32$-bits wide 16-MAC and 16-PAS-4-MAC for $B = 16$ weight bins; lower is better.

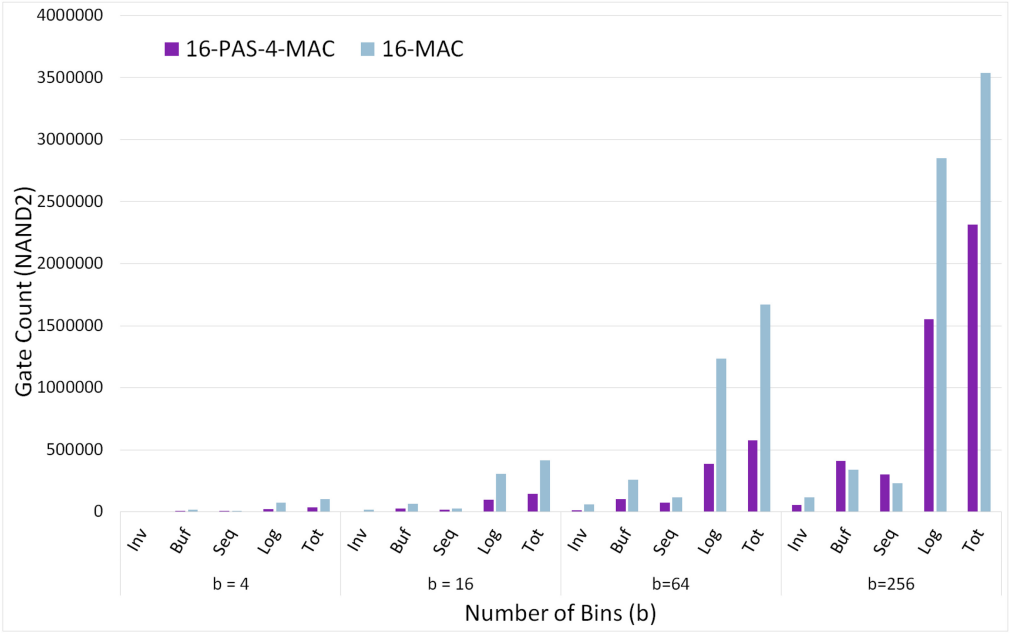


Fig. 9. Logic gate counts comparisons (in NAND2X1 gates) for $B = 4, 16, 64, 256$ weight bins for a 16-MAC and 16-PAS-4-MAC for $W = 32$-bit width; lower is better.

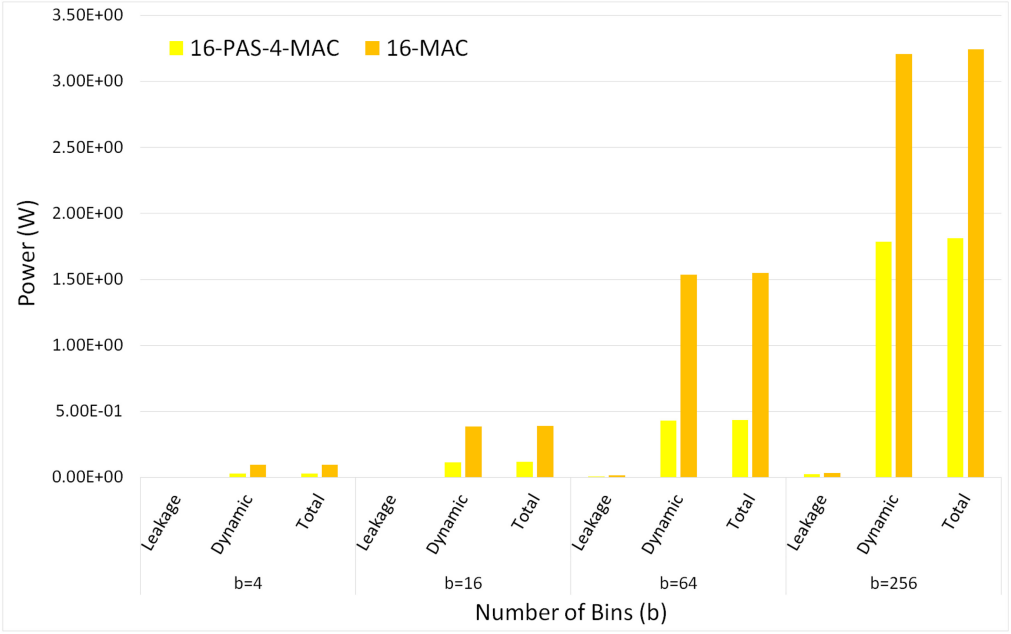


Fig. 10. Power consumption (in W) comparisons for $B = 4, 16, 64, 256$ weight bins deep 16-MAC and 16-PAS-4-MAC for $W = 32$-bit width; lower is better.

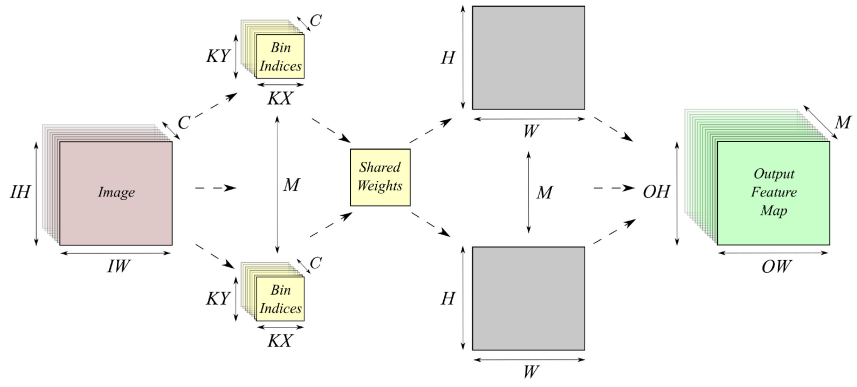


Fig. 11. Example of a aimplified weight-shared convolution.

accelerator? We attempt to answer this by implementing PASM in a weight-shared convolution layer accelerator and evaluate and compare its latency, power and area performance with a weight-shared convolution accelerator and baseline both against a non-weight shared convolution accelerator for the same clock speed. Figure 12 shows how, when PASM is implemented in a weight-shared convolution accelerator, multiple PAS units are created in parallel to accelerate the accumulation of $C \times IH \times IW$ ***image*** data into the corresponding B-bin registers. Multiplexers are created to expand and parallelize the ***image*** and ***binIndex*** data and demultiplexers then combine the PAS outputs for the post-pass MAC. The post-pass MAC multiplies and accumulates the binned ***image*** data with the corresponding $M \times C \times KX \times KY$ shared-weight value into the $M \times IH \times IW$ ***outFeat***.

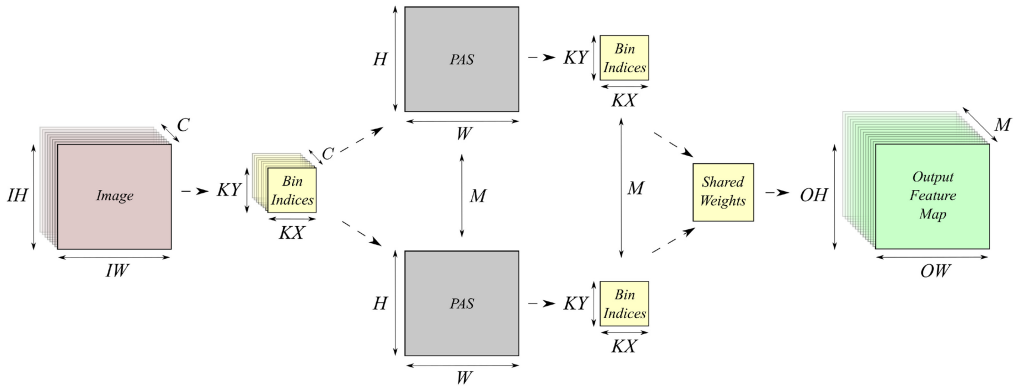


Fig. 12. Example of a simplified weight-shared convolution with PASM.

Table 2. Typical Numbers of MAC Operations

		input_channels (*C*)		
		32	**128**	**512**
kernels (*K*)	**1x1**	32	128	512
	3x3	288	1152	4608
	5x5	800	3200	12800
	7x7	1568	6272	25088

The ***image*** data of $C \times IH \times IW$ are buffered in registers, ***weight*** data of $M \times C \times KX \times KY$ are buffered in shared weight registers, the ***binIndex*** data up to 16 values are registered, and, finally, the output feature map of $M \times IH \times IW$ is registered in an ***outFeat*** register file. This allows for greater locality and reuse of the data.

As can be seen from Table 1 and Table 2, the PASM is only efficient when the number of PAS units created is much smaller than the number of items to accumulate, i.e., the PASM is efficient only where the number of bins, B, is much smaller that the number of pairs of inputs to be multiplied and summed, $C \times K \times K$. In the absence of quantization and weight-sharing, the PASM would not be viable. For example, if we tried to use PASM for 16-bit weight values without using quantization or weight-sharing, then we would need 2^{16} bins in the PASM. A PASM unit with so many bins would not be competitive with a conventional MAC unit.

Any weight-shared network such as a weight-shared AlexNet [14], weight-shared VGG [20] or weight-shared GoogLeNet [21], and more generally regional CNNs, recurrent neural networks (RNNs) and long short term memorys (LSTMs) are possible good candidates for the use of PASM, although the evaluation in these networks is beyond the scope of this article.

3.1 Examples

For a simplified weight-shared accelerator, Figure 11, each kernel channel is "slid" across the corresponding image channel, multiplying and accumulating each of the pixel values with the kernel's pre-trained weight-shared values into the corresponding interim feature map channel. Each of the interim feature map channels is then "stacked" to produce the output feature map.

Now assume a simplified weight-shared-with-PASM accelerator with the same number of channels and kernels, Figure 12. Again, each kernel channel is "slid" across the corresponding image channel; however, the "kernel" contains bin indices that address the interim feature map bin into

which the image pixel values are accumulated. After all the image channels have been accumulated into the image bins of the interim feature map, the bin indices are "slid" across the interim feature map, multiplying each of the accumulated image values with the corresponding kernel's indexed pre-trained weight-shared values, and accumulated into the associated output feature map channel.

Figure 13 shows the simplified SystemC code for weight-shared-with-PASM implemented within a convolution layer. It demonstrates an **image** of $C \times IH \times IW$, a ***kernel*** of $M \times C \times KY \times KX$, with B ***weight*** bins, a ***stride*** of S, and an ***outFeat*** of $M \times OH \times OW$.

4 DESIGN AND IMPLEMENTATION OF THE PASM CNN ACCELERATOR

For comparison, three versions of the accelerator, a non-weight-shared, a weight-shared, and a weight-shared-with-PASM accelerator, are designed and synthesized. The accelerators are coded in SystemC, which allows the designs to be partitioned, unrolled, and pipelined to optimize power and area (NAND2 equivalent gate count) by using SystemC *#pragma* directives rather than having to hand code the partitioning, unrolling, and pipelining in Verilog.

To increase the throughput of the PAS phase of the weight-shared-with-PASM CNN accelerator, the *imageBin* array of line 12 in Figure 13 is partitioned completely using the directive *ARRAY_PARTITION dim=1* (see line 2) to inform Xilinx Vivado_HLS to implement all bins in registers. When the *for* loop of line 9 to line 13 is unrolled using the directive *UNROLL* (see line 10) and loop merged using the directive *LOOP_MERGE* (see line 11), Vivado_HLS implements *imageBin* in registers rather than BRAM, allowing the high-level synthesis (HLS) to create multiple copies of the loop body so that it can parallelize the accumulation registers and associated accumulator logic and thus reduce the number of clock cycles of reads and writes to the *imageBin* registers.

The rest of the loops including the post pass MAC loop on lines 33 and 42 are pipelined with the directive *PIPELINE II = 1 rewind* that has an iteration interval of 1, suggesting to Vivado_HLS that the loops shall need to process a new input every cycle that Vivado_HLS will try to meet if possible. The *rewind* option is used with the pipeline function to enable continuous loop pipelining such that there is no pause between one loop iteration ending and the next beginning. This is effective as there are perfect nested loops in the convolution.

The partitioning, unrolling and loop merging reduces the latency cycles of the non-weight-shared, weight-shared and weight-shared-with-PASM accelerators by 92% at the expense of increasing the flip flop count by 97% and thus the power and area of these combined function and loop pipeline registers. Implementing the *imageBin* array in registers allows for cell compatibility in the ASIC synthesis tool and quick synthesis time as no static RAM (SRAM) needs to be modeled and implemented to store the input ***image*** and ***outFeat*** values. This increased power and area overhead of the accelerators is a good tradeoff for the increased throughput and lower latency.

The three versions of the CNN accelerators are based on the AlexNet [14] CNN and accelerate one layer of the convolution to allow for implementation in an FPGA. The accelerators include stride, an activation function, ReLU, and bias (a means for the network to learn more easily) as the activation function and bias parameters are not shared. Striding (lines 4, 5, and 42 of Figure 13) allows for compression of the image or input feature map by allowing differing pixel strides of the kernel across the input feature map. For a stride value of 1, the kernel is moved across the input feature map at a stride of one pixel at a time. With a stride of 2 or more the kernel jumps 2 or more pixels as the kernel strides across the feature map. This sliding of the kernels produces smaller spatial output feature maps. The use of PASM in the weight-shared accelerator is transparent to the functionality of the stride, activation function or biasing. Note that the numbers of weight parameters for a weight-shared system must be clustered (usually with k-means) and quantized to fit into 16- to 256-bins (see Han et al's. [10, 11] research), as this reduction in numbers of weights

```
bi[C][KY][KX], sk[B], bias[M], image[C][IH][IW], imageBin[B], outFeat[M][OH][OW];
#pragma HLS ARRAY_PARTITION variable=imageBin complete dim=1
#pragma HLS ALLOCATION instances=mul limit=1 operation
for (ihIdx=(KY/2); ihIdx<(IH-(KY/2)); ihIdx+=Stride) {
  for (iwIdx=(KX/2); iwIdx<(IW-(KX/2)); iwIdx+=Stride) {
    for (mIdx=0; mIdx<M; mIdx++) {
#pragma HLS PIPELINE II=1 rewind
// Reset the imageBin register file
      for (bin=0; bin<B;bin++) {
#pragma HLS UNROLL
#pragma HLS LOOP_MERGE
        imageBin[bin]=0;
      }

      binIdx=0;
      // For each channels, stride the kernel sized bin indices over the
      // image and accumulate the image value in the corresponding imageBin PAS
      for (cIdx=0; cIdx<C; cIdx++) {
        for (kyIdx=0; kyIdx<KY; kyIdx++) {
          for (kxIdx=0; kxIdx<KX; kxIdx++) {
            imVal=image[cIdx][((ihIdx+kyIdx)-(KY/2))][((iwIdx+kxIdx)-(KX/2))];
            binIdx=bi[cIdx][kyIdx][kxIdx];
            imageBin[binIdx] += imVal;
          }  // end for (kxIdx=0; ...
        }  // end for (kyIdx=0; ...
      }  // end for (cIdx=0; ...

      // Once looped over all the channels, stride the kernel sized bin indices
      // over the PAS and multiply with the corresponding shared-weight value.
      cIdx=0;
      for (kyIdx=0; kyIdx<KY; kyIdx++) {
        for (kxIdx=0; kxIdx<KX; kxIdx++) {
          mul[cIdx][kyIdx][kxIdx] =imageBin[bi[cIdx][kyIdx][kxIdx]] *
                                              sk[bi[cIdx][kyIdx][kxIdx]];
        }  // end for (kxIdx=0; ...
      }  // end for (kyIdx=0; ...

      // Sum all the channel's together into each ouptut feature map channel
      for (cIdx=0; cIdx<C; cIdx++) {
        for (kyIdx=0; kyIdx<KY; kyIdx++) {
          for (kxIdx=0; kxIdx<KX; kxIdx++) {
            outFeat[mIdx][ihIdx/Stride][iwIdx/Stride] += mul[cIdx][kyIdx][kxIdx];
          }  // end for (kxIdx=0; ...
        }  // end for for (kyIdx=0; ...
      }  // end for (cIdx=0; ...
    }  // for (mIdx=0; ...
  }  // for (iwIdx=(KX/2); ...
}  // for (ihIdx=(KY/2); ...
```

Fig. 13. Simplified System-C Code for the weight-shared-with-PASM convolution.

is what allows PASMs reduction in power and area by doing the PAS accumulations first followed by a single post pass MAC.

Our accelerators are high-level-synthesized to a hierarchical Verilog netlist using Xilinx Vivado_HLS (version 2017.1), which allowed for quick functional simulation and hardware co-simulation and could also allow for implementation in both ASIC and later FPGA. Vivado_HLS reports the approximate latency of the design along with the approximate utilization results for BRAM, DSP, flip flops, and look up tables (LUTs) after high-level synthesis has been executed.

When implementing the accelerators in FPGA, Xilinx Vivado (version 2017.1) is used to synthesize, optimize, place, and route the netlist from Xilinx Vivado_HLS into a Xilinx 7-series Zynq XC7Z045 FPGA part running at 200MHz. When implementing the accelerators into a 45nm process ASIC running at 1GHz, Cadence Genus is used to synthesize and optimize the design for ASIC. Cadence Genus supplies commands for reporting approximate timing, gate count and power consumption of the designs at the post-synthesis stage. The "report timing," "report gates," and "report power" commands of Cadence Genus are used to obtain the ASIC timing, gate count, and power results. The gate count is normalized to a NAND2 gate, and this number reported as the overall gate count.

The designs are coded using integer/fixed point precision numbers (INTs). The bit widths of the image are maintained at 32-bit INTs while the weights are stored as variable 8-bit, 16-bit, and 32-bit INTs. The bin indexes are stored as 2^2-bits for 4 weights up to 2^4-bits for 16 weights.

The encoding of finite state machines is set to gray encoding to keep the power consumption of the designs to a minimum. All registers and memories in the accelerators derived from variables in the SystemC are reset or initialized to zero. The resets are set as active low synchronous resets.

The number of kernels M is kept small, i.e., $M = 2$ to keep the synthesis time the ASIC tools to a minimum. The number of channels C is made as large as possible such that the $C \times KX \times KY$ is larger than B-bins to demonstrate the power saving effect of PASM compared to the same number of channels for the weight-shared version of the accelerator, as suggested in Table 1 and demonstrated in Table 2.

The *image* cache was kept to a small tile of the image of multiple channels ($IH = 5, IW = 5, C = 15$) to allow its implementation in a register file. However, the *image* cache could be implemented in SRAM in an ASIC. This would allow for a larger cache storage of image and weight values and further reduce the power and area of the accelerators but would require more "back-end" layout design work of the accelerator, something not considered for this article.[1] The *binIndex* would remain in a register file as a maximum of 16×32-bit values would be stored.

To further ensure the lowest number of multipliers utilized in the PASM accelerator the *ALLOCATION* directive is used to ensure that only one post pass multiplier is used further reducing the area and power while very slightly increasing the latency.

The Verilog netlists that are produced by Xilinx Vivado_HLS are synthesized for ASIC to produce a gate level netlist. Timing constraints in Synopsys design constraint (SDC) are created [7] so all versions of the accelerator meet timing at 1GHz with a short 0.01ns clock transition using Cadence Genus (version 17.11) synthesizer.

The synthesis targets the OSU FreePDK 45nm ASIC process cell library. Timing, latency, gate count (normalized to a NAND2 gate) and power consumption at different B-bins and W-bit widths

[1]The OSU FreePDK 45nm ASIC process cell library used for the experiments does not have a facility to synthesize on-chip SRAM in our implementation of the weight-shared-with-PASM accelerator. If we had access to a library that would allow SRAM synthesis, then we would be able to operate on larger data blocks in our ASIC design. The weight-shared-with-PASM is likely to be even more effective with larger input blocks (particularly a large value of C), because the cost of the post-pass multiplication can be amortized over more inputs.

are captured. These values are approximations as they are the post-synthesis estimates. The values will be optimized when implemented in ASIC or FPGA.

The weight-shared-with-PASM introduces a delay in processing the output of the PAS units. The PAS unit has a throughput of one pair of inputs per cycle, and so computes the initial accumulated values in about N cycles, where

$$N = (KX \times KY) \times C.$$

The post pass MAC unit also has a throughput of one pair of inputs per cycle, so requires one cycle for each of the B accumulator bins, for a total of $N + B$ PASM cycles. In contrast, a simple MAC unit requires just N cycles, however, consumes significantly more area and power, when compared to an accelerator with more than one PAS per MAC.

Table 2 shows the number of MAC operations that contribute to each output for various values of C and KX and KY. For example, if $C = 32$ input channels are used with kernels of dimensions $KX \times KY = 5 \times 5$, then each computed value will be the result of 800 MAC operations. A simple fully pipelined MAC unit might be able to compute this result in a little more than 800 cycles. As can be seen from lines 11 to 13 of Figure 1, each element of the output of a convolution layer of a CNN is the result of $C \times KX \times KY$ multiply-accumulate operations or 800 cycles in this example.

In contrast, a PASM has two phases: a PAS phase and a post-pass MAC phase. The PAS phase computes a histogram of the frequency of each weight input and depends entirely on the number of inputs. However, the post-pass MAC phase depends not on the number of inputs but on the number of different weights that can appear (each of which occupies one of the B-bins). Provided the number of inputs, $C \times KX \times KY$, is much larger than the number of bins, B, the cost of the post-pass remains small relative to the cost of the PAS phase. For example, if $B = 16$, then the cost of the post-pass will be a small fraction of the 800 operations needed at the PAS phase. Careful consideration of the size of bins used with respect to the number of channels and kernels is important due to the *summands* being multiplied-accumulated many times before the ***outFeat*** is updated as can be seen on lines 11–13 of Figure 1. The number of accumulations should therefore be much larger than B for PASM to be efficient in a weight-shared convolution accelerator.

5 EVALUATION OF PASM IN A CNN ACCELERATOR

5.1 ASIC Results

The PASM is implemented in a weight-shared CNN accelerator and synthesized into an ASIC. The latency is compared with that of the weight-shared accelerator. The latency results for each of the non-weight-shared, weight-shared, and weight-shared-with-PASM accelerators is obtained from Vivado_HLS Synthesis reports and the percentage differences graphed as seen in Figure 14. The latency of the weight-shared-with-PASM in Figure 14 was between 8.5% for 4-bin and 12.75% for 16-bin greater than that of the corresponding weight-shared version, which is expected due to the indirection of the PAS units.

Latency can be further reduced by relaxing the *ALLOCATION* directive (see line 3 of Figure 13) constraint on the multiplier. If more post-pass multipliers are used, then the latency drops with a corresponding increase in power and area, which may be acceptable depending on target device resources available.

For a 4-bin PASM accelerator, with 32-bit wide kernels, Figure 15(a) shows the gate count reports obtained from Cadence "report gates" command and normalized to a NAND2 gate. PASM uses 47.2% fewer total NAND2 gates compared with the non-weight-shared version and 47.8% fewer total NAND2 gates compared with weight-shared design. Figure 15(b) obtained from Cadence "report power" command, PASM uses 54.3% less total power when compared with its

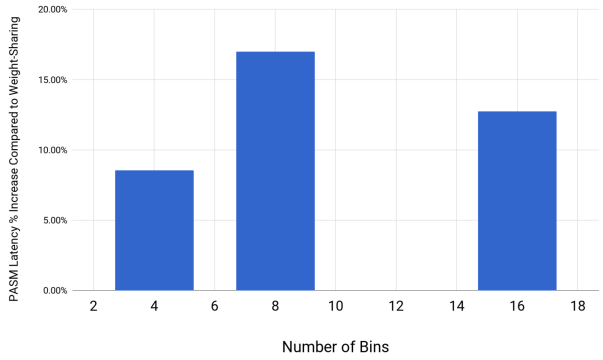
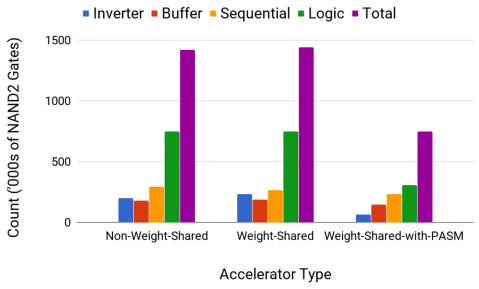
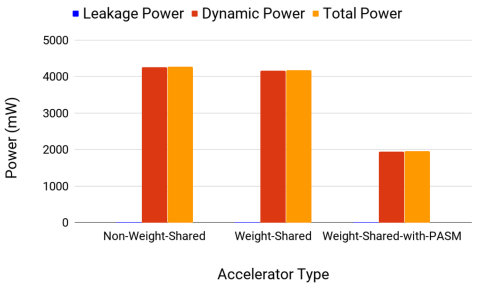


Fig. 14. Latency of weight-shared-with-PASM convolution compared to weight-shared convolution.

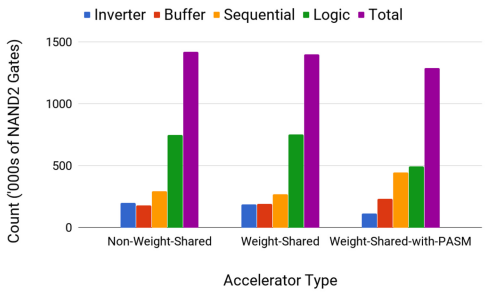


(a) ASIC Gate Count for 32-bit Kernel, 4-bin Accelerators

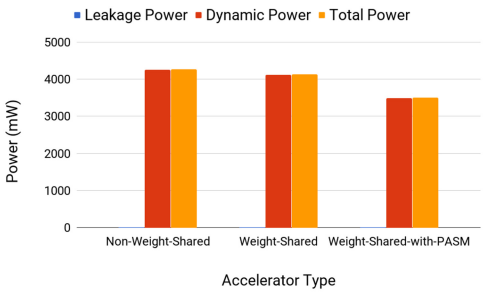


(b) ASIC Power Consumption for 32-bit Kernel, 4-bin Accelerators

Fig. 15. Four-bin, 32-bit kernel weight-shared-with-PASM vs. weight-shared gate count and power comparisons in ASIC.



(a) ASIC Gate Count for 32-bit Kernel, 8-bin Accelerators

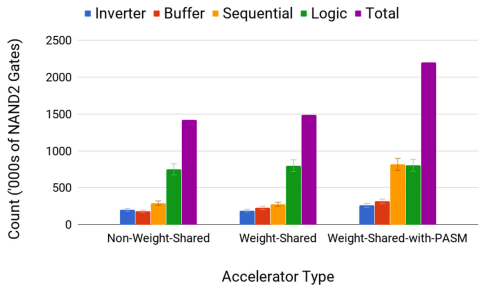


(b) ASIC Power Consumption for 32-bit Kernel, 8-bin Accelerators

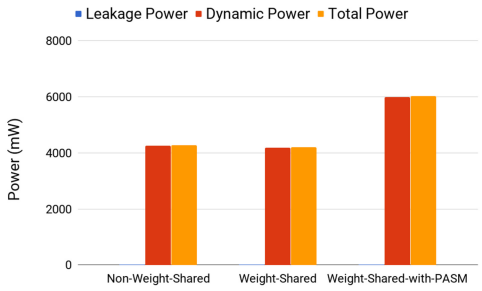
Fig. 16. Eight-bin, 32-bit kernel weight-shared-with-PASM vs. weight-shared gate count and power comparisons in ASIC.

non-weight-sharing counterpart and 53.2% less total power when compared with the weight-shared version.

For an 8-bin, 32-bit wide kernel PASM accelerator, Figure 16(a) obtained from Cadence "report gates" command and normalized to a NAND2 gate, PASM uses 9.4% fewer total NAND2 gates compared with the non-weight-shared and 8.1% fewer total NAND2 gates compared with the

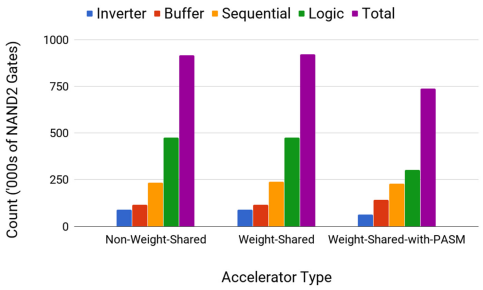


(a) ASIC Gate Count for 32-bit Kernel, 16-bin Accelerators

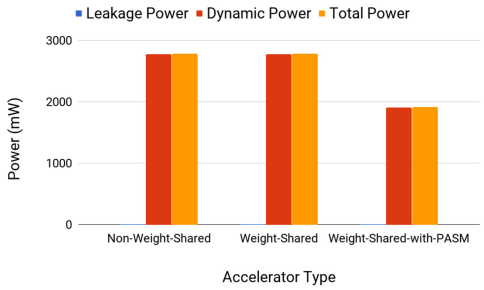


(b) ASIC Power Consumption for 32-bit Kernel, 16-bin Accelerators

Fig. 17. Sixteen-bin, 32-bit kernel weight-shared-with-PASM vs weight-shared gate count and power comparisons in ASIC.



(a) ASIC Gate Count for INT8-bit Kernel, 4-bin Accelerators



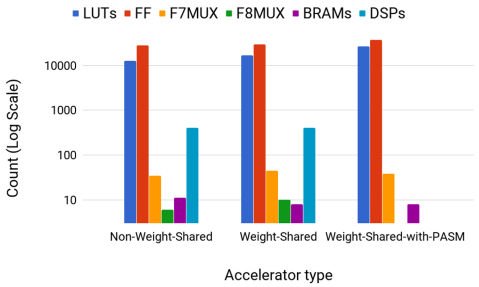
(b) ASIC Power Consumption for INT8-bit Kernel, 4-bin Accelerators

Fig. 18. Four-bin, INT8-bit kernel weight-shared-with-PASM vs. weight-shared gate count and power comparisons in ASIC.

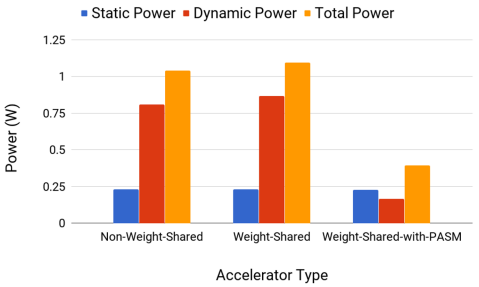
weight-shared accelerators. Figure 16(b) obtained from Cadence "report power" command, PASM consumes 18.1% less total power when compared with its non-weight-sharing and 15.2% less total power when compared with the weight-sharing accelerator.

For a 16-bin, 32-bit wide weight-shared-with-PASM accelerator, PASM no longer offers a good return with this level of unrolling, pipelining and partitioning of the ***imageBin***, at least when targeted at a 1GHz ASIC with this 45nm process cell library as it uses more NAND2 gates (see Figure 17(a)) and power (see Figure 17(b)) compared with the weight-shared accelerator. This is due to the ASIC tools having to increase the area and therefore power to meet timing at 1GHz for the 16-bins at 32-bit wide PASM. To achieve better power and area results for PASM at 16-bins or greater, it might be better to target a lower clock frequency, for example 800MHz. Alternatively, use a more efficient geometry ASIC cell library. Design changes could be made to reduce pipelining and unrolling of the levels of the inner four of the *for* loops of the convolutional code, which would reduce the area and power while making it easier for the ASIC tools to achieve timing, however, this may increase latency of the accelerator.

Due to the increased academic and industrial interest in applying INT8 approximations to reduce memory storage and bandwidth of the kernel data [3, 5], we show the results for the 8-bit kernel versions of the accelerators with 4-bins. This demonstrates that for a bin depth of 4, PASM achieves a 19.8% reduction in gate count, Figure 18(a) obtained from Cadence "report gates" command and



(a) Cell Count for 32-bit Kernel, 4-bin Accelerators



(b) Power Consumption for 32-bit Kernel, 4-bin Accelerators

Fig. 19. Four-bin, 32-bit kernel weight-shared-with-PASM vs. weight-shared gate count and power comparisons in FPGA.

normalized to a NAND2 gate and a 31.3% reduction in power compared to the weight-sharing version, and Figure 18(b) obtained from Cadence "report power" command.

5.2 FPGA Results

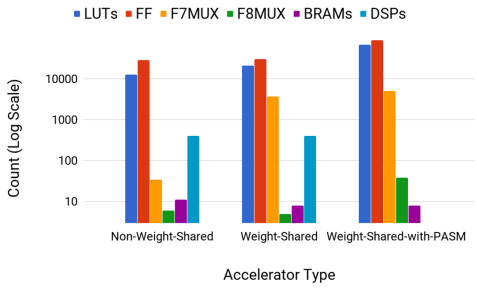
We implement the weight-shared-with-PASM accelerator in the Xilinx 7-series Zynq FPGA, the XC7Z045 part implemented on the Zynq ZC706 development board. Timing constraints in Xilinx design constraint (XDC) are created such that the accelerator designs met timing at 200MHz. The resets are set as active high synchronous resets for better FPGA power performance. The state machines are set to gray encoding.

The ***image***, ***imageBin***, and ***kernel*** were cached in BRAM in the FPGA. This allows for a larger cache storage of image and weight values and further reduce the power and area of the accelerator. However, a larger image and kernel cache could be employed for greater throughput of the accelerators but for the purposes of comparison with the ASIC implementation, the same image and kernel dimensions are used.

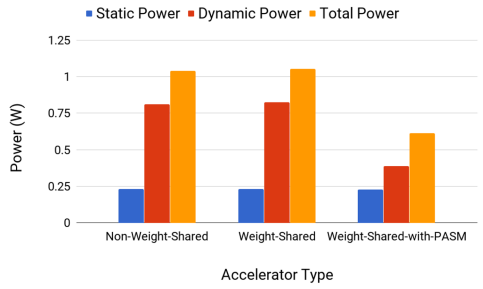
When using the *UNROLL* and *PIPELINE* directives with the *for* loops and using Vivado_HLS synthesis followed by RTL synthesizing and fully implementing the designs with Vivado, the non-weight shared and weight-shared versions of the 16-bin, 32-bit kernel data designs utilizes 405 DSP units on the FPGA of the ZC706 board. If a smaller, more resource constrained FPGA is required for cost reasons, like the Xilinx XC7Z020 part found on the Xilinx PYNQ-Z1 low cost development board, then the non-weight shared and weight-shared versions of the design would over utilize the 220 DSP units of the PYNQ-Z1 board's XC7Z020 FPGA part.

The weight-shared-with-PASM version of the design for the same 4-bin, 32-bit kernel, Figure 19(a) obtained with Vivado's "report_utilization" command, similarly unrolled and pipelined, HLS synthesized in Vivado_HLS followed by RTL synthesized and fully implemented in Vivado, only utilizes 3 DSP units, 99% fewer DSPs than the other versions of the accelerator with the same 12% increase in latency as the ASIC implementation. PASM also consumes 28% fewer BRAMs, while consuming 64% less power than the weight-shared accelerator, Figure 19(b), obtained with Vivado's "report_power" command. Increasing the number of post-pass MACs decreases the latency slightly while increasing the power consumption and DSP usage, as the bottleneck is in the accumulators of the PAS that can be seen as the number of $C \times KX \times KY$ accumulations that must be larger than that of B-bins for PASM to be effective and efficient as seen in Table 1 and Table 2.

For an 8-bin PASM accelerator, with 32-bit kernels, Figure 20(a) obtained with Vivado's "report_utilization" command, PASM uses 99% fewer DSPs and 28% fewer BRAMs compared with the

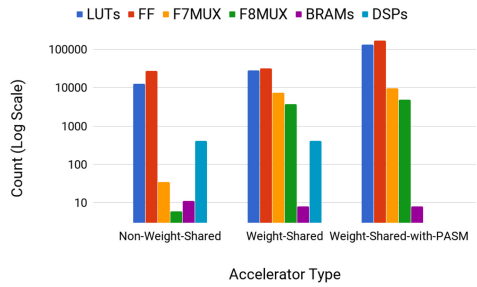


(a) Cell Count for 32-bit Kernel, 8-bin Accelerators

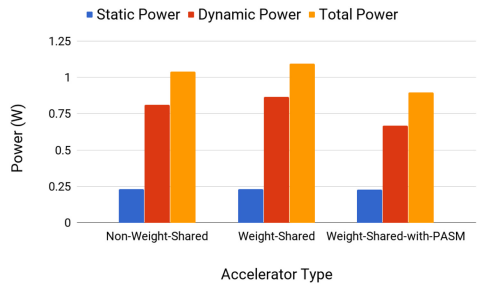


(b) Power Consumption for 32-bit Kernel, 8-bin Accelerators

Fig. 20. Eight-bin, 32-bit kernel weight-shared-with-PASM vs. weight-shared gate count and power comparisons in FPGA.



(a) Cell Count for 32-bit Kernel, 16-bin Accelerators



(b) Power Consumption for 32-bit Kernel, 16-bin Accelerators

Fig. 21. Sixteen-bin, 32-bit kernel weight-shared-with-PASM vs. weight-shared gate count and power comparisons in FPGA.

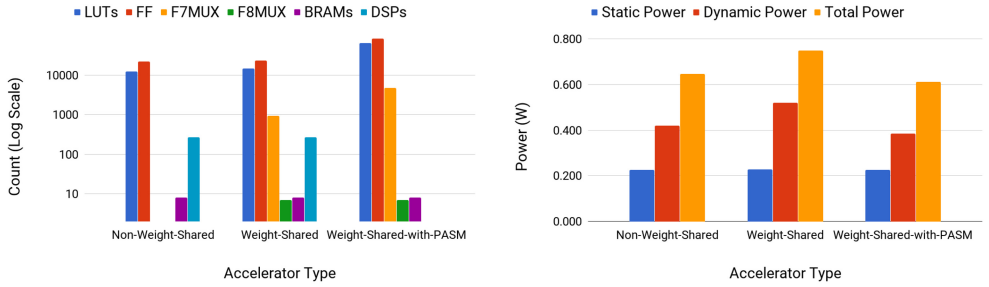
weight-shared design. Figure 20(b) obtained with Vivado's "report_power" command, PASM uses 41.6% less total power when compared with its weight-shared version.

For a 16-bin PASM accelerator, with 32-bit kernels the utilization reported with Vivado's "report_utilization" command, Figure 21(a), PASM uses 99% fewer DSPs and 28% fewer BRAMs compared with the weight-shared design. Figure 21(b), PASM uses 18% less total power when compared with its weight-shared version, reported with Vivado's "report_power" command.

It is also possible to clock the PASM at higher clock speeds for the same latency than that of the weight-shared counterpart, but again for the sake of comparison, clock speeds are kept consistent between all versions of the accelerators.

If INT8 approximations are desired for the weight data, then an 8-bit wide, 8-bin PASM accelerator, Figure 22(a) again obtained with Vivado's "report_utilization" command, uses 99% fewer DSPs but the same number of BRAMs as its weight-shared counterpart. In Figure 22(b), PASM uses 18.3% less total power when compared with its weight-shared version.

For a 16-bin, 8-bit wide PASM accelerator, PASM no longer offers a good return when targeted at a 200MHz FPGA with this level of unrolling, pipelining and partitioning of the ***imageBin*** as it uses more flip-flop gates and power, exceeding the gate count and power of the DSP units being used in the weight-shared accelerator. At this stage, it would be better to either implement the



(a) Cell Count for INT8-bit Kernel, 8-bin Accelerators (b) Power Consumption for INT8-bit Kernel, 8-bin Accelerators

Fig. 22. Eight-bin, INT8-bit kernel weight-shared-with-PASM vs. weight-shared gate count and power comparisons in FPGA.

imageBin in dual port BRAM and incur a slight increase in latency or do not unroll and pipeline as many levels of the inner four of the *for* loops of the convolutional code.

5.3 Overall Results

The precision of the results of a weight-shared CNN accelerator that uses PASM are identical to that of a weight-shared CNN accelerator using traditional MACs. The same filters and image data are being used for the weight-shared accelerator as demonstrated in Figure 4 and the weight-shared-with-PASM accelerator shown in Figure 6. While PASM has a different underlying process of permuting the convolution, the results of a convolution layer are identical to that of a standard MAC weight-shared accelerator, except PASM adds a 12.5% increase in latency in obtaining the result but with vastly reduced power consumption and area (NAND2 gates) compared to the traditional MAC version.

As suggested in Han et al. [10], they show that the Top-5 classification accuracy of their weight-shared CNN accelerator is 19.70% compared to 19.73% Top-5 accuracy of the baseline non-weight-shared CNN accelerator due to there being many less filter weight values. When PASM is used in a weight-shared CNN accelerator the classification accuracy is unaffected when compared to the baseline weight-shared CNN accelerator counterpart as the same filter weight values of the weight-shared CNN accelerator are used and the same output feature map results are obtained.

PASM is beneficial for up to 16 weight bins and 32-bits for FPGA at 200MHz and 8 weight bins and 32-bits for ASIC at 1GHz 45nm process when coded using SystemC with the above unrolling, pipelining and partitioning configuration. As demonstrated earlier in the article, were a weight-shared-with-PASM CNN accelerator to be coded in Verilog, the numbers of bins supported could indeed be higher. We wanted to experiment with differing pipelining, unrolling and partitioning directives and their effect on making PASM more efficient, something that would have been impractical had it been coded in Verilog, so SystemC was used. Further SystemC and other SRAM optimizations (for image and output feature map caching) could have been done to the accelerators, but this was not the focus of this article and may be undertaken as future work.

6 RELATED WORK

There have been many different CNN hardware accelerators proposed for both FPGA and ASIC. Gupta et al. [9] show increased efficiency in an FPGA hardware accelerator of a 16-bit fixed-point representation using stochastic rounding without loss of accuracy. Zhang et al. [22] deduced the best CNN accelerator taking FPGA requirements into consideration and then implement the best

on an FPGA to demonstrate high performance and throughput. Chen et al. [1] design an ASIC accelerator for large-scale CNNs focusing on the impact of memory on the accelerator performance.

Han et al. [10] have proposed an Efficient Inference Engine that builds on their 'Deep compression' [11] work to perform inferences on the deeply compressed network to accelerate the weight-shared matrix-vector multiplication. This accelerates the classification task while saving energy when compared to CPU or GPU implementations. Given that one aspect of deep compression is quantizing and dictionary encoding weights, we believe that the use of our PASM units might further reduce resource and energy requirements.

Chen et al. [2] address the problem of data movement that consumes large amounts of bandwidth and energy in their *Eyeriss* accelerator. They focus on data flow in the CNN to minimize data movement by reusing weights within the hardware accelerator to improve locality. This was implemented in ASIC and power and implementation results compared showing the effectiveness of weight reuse in saving power and increasing locality. Chen et al. reduce the required memory bandwidth primarily be reusing data that is already on-chip rather through weight compression. However, the two approaches are mostly orthogonal, so our PASM approach could potentially work together with an *Eyeriss* type accelerator.

Ma et al. [17] present an in-depth analysis of convolution loop acceleration strategies by numerically characterizing the loop optimization techniques. They do this by looking at different levels of loop unrolling and loop tiling (subdividing the design into smaller blocks) and loop interchange (different ordering of the loops). They also consider latency and partial sum storage and how they can minimize both. They provide design guidelines for an efficient implementation of the accelerator to minimize latency, minimize partial sum storage, minimize both on-chip buffer accesses and off-chip memory accesses. For the four inner convolution loops, they show that loops to unroll (in this case all four loops), which to tile (loops 1 and 2 are buffered), and which to interchange (compute loop 1 then loop 2 but it doesn't matter the order of loop 3 and loop 4). They implement the accelerator for a VGG-16 CNN model in an Arria-10 GX 1150 FPGA (3600 DSPs, 18×18 20kb random access memory (RAMs)) at 150MHz and coded in Verilog achieving 645.25 giga operations per second (GOPS) of throughput and 47.97ms of latency per image. This work on loop ordering is complementary to our work on architecting a lower-resource MAC unit.

Several research groups have studied the effects of lower precision weight values for CNNs. Reducing the data precision of weight data is an alternative method of quantizing that is different to the weight sharing of Han et al. [10]. Rather than selecting a set of quantized values guided by the values in the data, low-precision approaches simply quantize existing weights to the nearest low-precision value. A particularly popular data type is 8-bit integers.

Dettmers [3] shows how 8-bit for data and model parallelism increases the performance of machine learning while maintaining accuracy on MNIST, CIFAR10 and ImageNet neural networks. The article describes data parallelism across multiple GPUs showing bandwidth and latency limitations on the peripheral component interconnect express (PCIe). They show different ways of representing the mantissa and exponent in the available 8-bit. They show how the 32-bit value is compressed into the 8-bit value and decompressed. They show how representing the 8-bit using a dynamic tree data type is able to approximate random numbers better than other known data types but interestingly all approximation techniques (dynamic tree, linear quantization, 8-bit mantissa and static tree) work well in training. They investigate other sub 32-bit data types and show that model parallelism in conjunction with sub-batches works very well in networks and avoids the problem of large batch sizes for 1-bit quantization proposed by Seide et al. [19].

In the Xilinx white paper of Fu et al. [5], Xilinx makes use of 18-bit and 27-bit hardware multipliers that are commonly found on Xilinx FPGAs. They use these multipliers to compute two 8-bit

multiplications in parallel, giving better performance and efficiency than if each multiplier were to perform just one 8-bit multiplication per cycle. This approach is quite different to our proposal for a new type of MAC unit, and depends on the presence of existing hard-coded multipliers on the FPGA.

7 CONCLUSION

ASICs and FPGAs are often used to hardware accelerate the convolution layers of a CNN where up to 90% of the computation time is consumed. This computation requires large amounts of multipliers as part of the many thousands of MAC operations needed in the convolution layer. These multipliers consume large amounts of physical and computational IC die resources or DSP units on a FPGA. Hardware accelerators have been proposed that reduced the amount of kernel data required by the neural network by dictionary compressing the weight values after training the network. This "weight sharing" reduces the bandwidth and power of the data transfers from external memory but still requires large numbers MAC units.

We reduce power and area of the CNN accelerator by implementing PASM in a weight-shared CNN accelerator. PASM re-architects the MAC to instead count the frequency of each weight and place it in a bin. The accumulated value is computed in a subsequent multiply phase, significantly reducing gate count and power consumption of the CNN. We coded in Verilog a 16-MAC weight-shared accelerator and a 16-PAS-4-MAC weight-shared-with-PASM accelerator and compare the logic resource requirements of a $b = 16$ bin for varying w-bit widths. Gate counts are normalized to a NAND2X1 gate. For $w = 32$-bit wide the 16-PAS-4-MAC has overall 66% fewer logic gates and consumes 70% less total power than the 16-MAC.

To further evaluate the efficiency gains of PASM, we implement PASM in a weight-sharing CNN accelerator. We compare it to a non-weight-shared accelerator and a weight-shared accelerator, targeted at a 1GHz 45nm ASIC. The gate count area and power consumption for the weight-shared-with-PASM is lower compared to the weight-shared version. For a 4-bin weight-shared-with-PASM accelerator that accepts a 5×5 image with a 3×3 kernel and 15 input channels and 2 output channels, an ASIC implementation of PASM saves 48% NAND2 gates and 53.2% power when compared to its weight-shared counterpart, with only a 12% increase in latency.

We show that the weight-shared-with-PASM accelerator can be implemented in a resource-constrained FPGA. For an accelerator with the same dimensions as the ASIC version implemented on the FPGA to run at 200MHz, PASM uses 99% fewer DSPs and 28% fewer BRAMs compared with the weight-shared design. For 16-bin PASM, PASM uses 18% less total power when compared with its weight-shared version.

Even if INT8 approximations are desired for the weight data, an 8-bit wide, 4-bin PASM accelerator running at 200MHz on the FPGA uses 99% fewer DSPs and 28% fewer BRAMs compared with the weight-shared design. An INT8 operation PASM also uses 47% less total power when compared with its INT8 weight-shared version.

Quantization and weight-sharing neural networks are active research areas, particularly for reducing DRAM bus bandwidth usage and in applications such as RNNs and LSTM networks. Weight sharing allows for implementation of a CNN in small, low power embedded systems as less RAM is required to store the weight values. Weight sharing also offers a more rapid way of implementing an inference network on a small memory embedded device without the large training phase required of, say a Binary Neural Network. Weight sharing is used in other types of networks such as regional-CNNs, RNNs and LSTMs so PASM may be a good fit there too. Wherever the number of shared weights is sufficiently small, PASM units may be an attractive alternative to a conventional weight-sharing MAC unit.

ACKNOWLEDGMENTS

We also extend our thanks and appreciation to the Institute of Technology Carlow, Carlow, Ireland for their support.

REFERENCES

[1] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the 19th International Conference on Architectural Support Programming Languages and Operating Systems*. 269–284. DOI: http://dx.doi.org/10.1145/2541940.2541967

[2] Yu Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Proceedings of the 2016 43rd International Symposium on Computer Architecture (ISCA'16)*. IEEE, 367–379. DOI: http://dx.doi.org/10.1109/ISCA.2016.40

[3] Tim Dettmers. 2016. 8-bit approximations for parallelism in deep learning. In *4th International Conference on Learning Representations (ICLR'16)*. San Juan, Puerto Rico.

[4] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello. 2010. Hardware accelerated convolutional neural networks for synthetic vision systems. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. 257–260. DOI: http://dx.doi.org/10.1109/ISCAS.2010.5537908

[5] Yao Fu, Ephrem Wu, Ashish Sirasao, Sedny Attia, Kamran Khan, and Ralph Wittig. 2016. Deep learning with INT8 optimization on Xilinx devices white paper (WP485). 486, WP486 (v1.0.1) (2016), 1–11.

[6] Martin Fürer. 2007. Faster integer multiplication. In *Proceedings of the 39th Annual ACM Symposium on the Theory of Computing*. 57–66. DOI: http://dx.doi.org/10.1145/1250790.1250800

[7] Sridhar Gangadharan and Sanjay Churiwala. 2015. *Constraining Designs for Synthesis and Timing Analysis: A Practical Guide to Synopsys Design Constraints (SDC)*. Springer.

[8] J. Garland and D. Gregg. 2017. Low complexity multiply accumulate unit for weight-sharing convolutional neural networks. *IEEE Comput. Arch. Lett.* 16, 2 (Jul. 2017), 132–135. DOI: http://dx.doi.org/10.1109/LCA.2017.2656880

[9] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. In *Proceedings of the 32nd International Conference on Machine Learning*. 1737–1746.

[10] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. 2016. EIE: Efficient inference engine on compressed deep neural network. In *Proceedings of the 43rd International Symposium on Computer Architecture*. 243–254. DOI: http://dx.doi.org/10.1109/ISCA.2016.30

[11] Song Han, Huizi Mao, and William J. Dally. 2016. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations (ICLR'16)*.

[12] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*. 770–778. DOI: http://dx.doi.org/10.1109/CVPR.2016.90

[13] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Sign. Process. Mag.* 29, 6 (Nov. 2012), 82–97. DOI: http://dx.doi.org/10.1109/MSP.2012.2205597

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems—Volume 1 (NIPS'12)*. Curran Associates Inc., 1097–1105. http://dl.acm.org/citation.cfm?id=2999134.2999257

[15] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural Comput.* 1, 4 (Dec. 1989), 541–551. DOI: http://dx.doi.org/10.1162/neco.1989.1.4.541

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (Nov. 1998), 2278–2324. DOI: http://dx.doi.org/10.1109/5.726791

[17] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae-sun Seo. 2017. Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'17)*. 45–54. DOI: http://dx.doi.org/10.1145/3020078.3021736

[18] S. Sabeetha, J. Ajayan, S. Shriram, K. Vivek, and V. Rajesh. 2015. A study of performance comparison of digital multipliers using 22nm strained silicon technology. In *Proceedings of the 2015 2nd International Conference on Electronics and Communication Systems (ICECS'15)*. 180–184. DOI: http://dx.doi.org/10.1109/ECS.2015.7124888

[19] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 2014. 1-bit stochastic gradient descent and application to data-parallel distributed training of speech DNNs. In *Proceedings of the Annual Conference of the International Speech Communication Association (Interspeech'14)*.

[20] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *CoRR* abs/1409.1556 (2014). arXiv:1409.1556 http://arxiv.org/abs/1409.1556

[21] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 1–9. DOI: https://doi.org/10.1109/CVPR.2015.7298594

[22] 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 *Proceedings of the ACM/SIGDA International Symposium on the FPGAs*. 161–170. DOI: http://dx.doi.org/10.1145/2684746.2689060

Received January 2018; revised May 2018; accepted June 2018