Received 25 February 2020; revised 14 May 2020 and 16 June 2020; accepted 30 June 2020. Date of publication 7 July 2020; date of current version 26 August 2020. This article was recommended by Guest Editor X. Zhang.

Digital Object Identifier 10.1109/OJCAS.2020.3007334

Power Efficient Tiny Yolo CNN Using Reduced Hardware Resources Based on Booth Multiplier and WALLACE Tree Adders

FASIH UD DIN FARRUKH (Graduate Student Member, IEEE), CHUN ZHANG (Member, IEEE), YANCAO JIANG, ZHONGHAN ZHANG, ZIQIANG WANG (Member, IEEE), ZHIHUA WANG (Fellow, IEEE), AND HANJUN JIANG (Senior Member, IEEE)

Institute of Microelectronics, Tsinghua University, Beijing 100084, China
Tsinghua National Laboratory for Information Science and Technology, Beijing 100084, China

CORRESPONDING AUTHOR: H. JIANG (e-mail: jianghanjun@tsinghua.edu.cn)

This work was supported by National Science Technology Major Project under Grant 2016ZX01012101.

ABSTRACT Convolutional Neural Network (CNN) has attained high accuracy and it has been widely employed in image recognition tasks. In recent times, deep learning-based modern applications are evolving and it poses a challenge in research and development of hardware implementation. Therefore, hardware optimization for efficient accelerator design of CNN remains a challenging task. A key component of the accelerator design is a processing element (PE) that implements the convolution operation. To reduce the amount of hardware resources and power consumption, this article provides a new processing element design as an alternate solution for hardware implementation. Modified BOOTH encoding (MBE) multiplier and WALLACE tree-based adders are proposed to replace bulky MAC units and typical adder tree respectively. The proposed CNN accelerator design is tested on Zynq-706 FPGA board which achieves a throughput of 87.03 GOP/s for Tiny-YOLO-v2 architecture. The proposed design allows to reduce hardware costs by 24.5% achieving a power efficiency of 61.64 GOP/s/W that outperforms the previous designs.

INDEX TERMS Convolutional neural network, booth encoding multiplier, WALLACE tree adders, FPGA, adder tree, object detection.

I. INTRODUCTION

DEEP learning evolves from machine learning and it is quickly becoming an essential part of daily life. A deep convolutional neural network is a part of deep learning and it facilitates to resolve many complex image-related tasks [1]–[3]. It has been successfully applied in a wide range of applications that include classification, speech processing and recognition, and object detection [4]–[6]. Moreover, deep learning is also becoming a potential solution for many industrial applications. These applications include autonomous vehicles, smart robots and camera technologies, and surveillance [7]–[10]. GPUs, FPGAs, and ASICs are used to implement the CNN accelerator design. GPUs have the advantage of design flexibility, but are energy inefficient and usually require a long execution time. The ASICs consume less power than the GPUs, but the flexibility is sacrificed, and the implementation cycle is quite long in consideration of the chip fabrication. In comparison with GPUs and ASICs, FPGAs have a good trade-off in terms of design flexibility, the implementation cycle, and the power consumption. FPGAs can be reconfigured depending on the application requirement. The FPGA designs can also be easily converted to ASIC designs. In recent times, the benefits of FPGAs in energy-efficiency, reconfigurable architecture, and customizable features draw the attention of many researchers to put their focus on FPGA based accelerator design. However, it remains a challenge to develop a hardware design of CNN accelerator for energy and area efficient systems.

FPGA becomes a suitable candidate as compared to GPU to implement the CNN accelerator [11]–[15]. However, FPGA has a limited number of hardware resources like MAC units and on-chip memories. Therefore, there is a need to find an efficient method to overcome these problems. Over the years, optimizing the CNN design on FPGA has been presented by many researchers [16]–[18]. Limited MAC units on FPGA caused a problem to perform convolution [14]. If a direct hardware mapping of CNN on FPGA is required, DSP blocks became the bottleneck [19]. To implement convolution on FPGA, multi-operand adders are equally important as much as MACs. Consider the direct hardware mapping on FPGA, 69% of the logic is consumed in first convolution layer by multi-operand adders [20]. To overcome the limitations of MACs, an optimization strategy based on the WALLACE tree multiplier was proposed as an alternate solution to power and area hungry MAC units [21]. Time-multiplexed serialization and approximate computing techniques were employed to counter multi-operand adders (MOAs) but failed to achieve expected results [20]. WALLACE tree adders provided an alternate solution to MOAs [22].

The CNN structure typically contains multiple operations. However, convolution is the most expensive computation engine of CNN in order to implement on hardware like FPGA. Its complexity can be determined by the fact that more than 90% of the computational time is consumed by convolution [23]. Convolution process is based on the 3-D multiplication and addition of input feature maps (or channels) N_c with N_k convolution kernels $K_x \times K_y$ and it can be represented as (1) [22]:

$$O(n,u,v) = \sum_{i}^{N_c}\sum_{x}^{K_x}\sum_{y}^{K_y} I(i, s\times u + x, s\times v + y) \times W(n,i,x,y) \tag{1}$$

The output feature map is obtained after convolution (dot-product) of the input feature map with the kernel weights and multipliers are used to perform this task. After multiplication, the outputs are given to the next stage of the adder tree to perform the addition on intermediate results. The pseudocode for the convolutional layer is shown in code 1. From pseudocode 1, it is evident that convolution consists of four levels of loops that slide along both kernel and feature maps. Therefore, it resulted in large design space to find a solution to implement parallelism, sequencing of computations, and divide the large data into smaller data sets to accommodate into built-in memory. Loop optimization techniques such as loop unrolling, tiling, and interchange helped to handle these problems [18].

The basic entity of the CNN accelerator is a processing element (PE) that performs the convolution task. In this work, a modified Booth encoding (MBE) algorithm is proposed

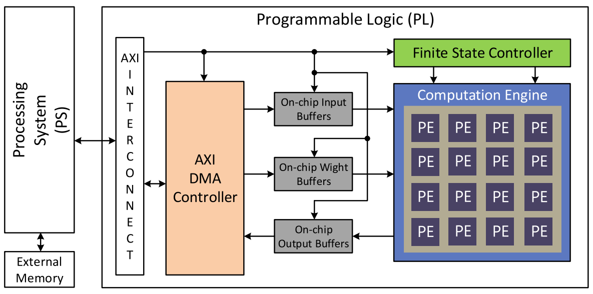

FIGURE 1. System Architecture.

Code 1 The Pseudocode for a Convolutional Layer

```
1 for($r = 0; r < R; r++$) {       } ——→ Loop-4
2   for($c = 0; c < C; c++$) {     }
3     for ($no = 0; no < M; no++$) { ——→ Loop-3
4       for ($ni = 0; ni < N; ni++$) { ——→ Loop-2
5         for ($x = 0; x < K; x++$) {     } ——→ Loop-1
6           for ($y = 0; y < K; y++$) {   }
              out_fmap[$no$][$r$][$c$] += weight[$no$][$ni$][$x$][$y$]*
              in_fmap[$ni$][S*$r + x$][S*$c + y$];
  }}}}}}
```

and implemented to perform multiplication. Similarly, a deep binary adder tree is replaced by the WALLACE tree-based adders. The key benefits to achieve from proposed design are high power efficiency and low hardware cost. Therefore, the proposed PE based on MBE multiplier with WALLACE tree adders has the advantage of low power consumption with the reduced hardware overhead. The main contributions of this work are summarized as follows:

1. Due to constraints of computational resources, optimization for CNN accelerator design is performed based on uniform loop unrolling and tiling for convolution layers.
2. Replacing the MAC unit with an MBE multiplier to overcome the problem of bulky MACs. The different designs of MBE are implemented:
   a. With a sign extension logic in the generation of partial products.
   b. To overcome the challenge of sign extension, sign extension elimination is applied. As a result, a correction vector is generated and it helps to save hardware resources that are occupied by sign extension logic.
   c. WALLACE reduction using carry-save adders (CSAs) are designed to reduce the partial products.
3. WALLACE tree-based adders are proposed to replace the MOAs that consume most of the logic and area.

The proposed architecture is implemented and tested for object detection task and achieves the power efficiency of 61.64 GOP/s/W. The LUTs consumption of proposed PE unit is reduced by 29.5%, power consumption is improved to 22.1%, and the overall system's hardware reduction attained by 24.5% that outperforms the previous approaches.

Code 2 The Pseudocode for a Proposed Accelerator Design

```
for ($r = 0$; $r <$ R; $r+=$ Tr) {
  for ($c = 0$; $c <$ C; $c+=$ Tc) {
    for ($no = 0$; $no <$ M; $no+=$ Tm) {
      for ($ni = 0$; $ni <$ N; $ni+=$ Tn) {
------------------------------------------------------------
for ($x = 0$; $x <$ K; $x++$) {          On-Chip data computation
  for ($y = 0$; $y <$ K; $y++$) {
    for ($nrr = r$; $nrr < \min(r+$Tr, R$)$; $nrr++$) {
      for ($ncc = c$; $ncc < \min(c+$Tc, C$)$; $ncc++$) {
        for ($noo = no$; $noo < \min(no+$Tm, M$)$; $noo++$) {
          for ($nii = ni$; $nii < \min(ni+$Tn, N$)$; $nii++$) {
            out_fmap[$noo$][$nrr$][$ncc$] += weight[$noo$][$nii$][$x$][$y$]*
            in_fmap[$nii$][S*$nrr$ + $x$][S*$ncc$ + $y$];
  }}}}}}}}}}
```

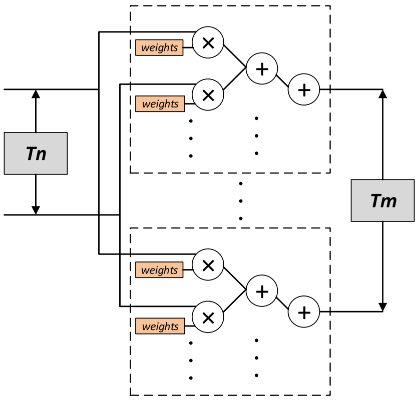

FIGURE 2. The computation engine of CNN.

The rest of the paper is organized as follows. In Section II, the proposed CNN accelerator design overview, memory organization, and prior works on multipliers are presented. The proposed PE unit based on MBE multiplier with WALLACE tree-based adders is described in Section III. Experimental results and discussion are in Section IV followed by a conclusion in Section V.

II. SYSTEM ARCHITECTURE AND OVERVIEW

The accelerator design for the inference phase of CNN consists of computation engine, on-chip memory buffers, and off-chip memory. Computation engine contains processing elements (PEs) and it is the main core of the CNN accelerator to perform convolution task. The required data for PEs are stored to on-chip memories after fetching from the external memory.

A. ACCELERATOR DESIGN OVERVIEW

The system architecture of the CNN accelerator is shown in Fig. 1. A CNN accelerator design on FPGA consists of several major components like computation engine, on-chip buffers, interconnects, memory controllers, and off-chip memory. The data need to be processed are stored in external memory. There are limitations of on-chip memory and therefore, data is first stored in on-chip buffers and then transferred to the PEs. In this work, optimization is performed for a computation engine of accelerator design and an optimized PE is implemented and used for a proposed design.

B. PROCESSING ELEMENT

The main computation part of CNN accelerator design is PE. In PE, convolution is performed on the input feature map with the shifted window of $K \times K$ kernel to generate one pixel in an output feature map. For CNN accelerator design, there are many potential solutions that can be explored to reduce the hardware implementation cost. Loop transformation techniques like loop unrolling, loop tiling, and loop interchange are used to optimize the computation of convolution layer to efficiently use the FPGA hardware resources [18].

Due to limited resources on FPGA, the whole convolution cannot be performed at once. Therefore, an idea of loop unrolling is employed to increase the utilization of FPGA computational resources. With the idea of loop tiling, convolution is cut into slices and each slice performing the convolution [14], [18]. Therefore, it helps in effectively using the limited on-chip memory because FPGA has a limited memory to store the data. Loop interchange determines the computation order of the four loops as shown in pseudocode 1. There are two types of loop interchange technique called inter-tile and intra-tile. Inter-tile loop order determines that how data moves from external memory to on-chip and intra-tile tells how data moves from on-chip buffers to PEs. The pseudocode of the proposed accelerator design is shown in code 2. Similarly, tile sizes for pseudocode 2 are given below:

$$\left.\begin{array}{l} 0 < Tn \times Tm \leq (\#\ of\ PEs) \\ 0 < Tm \leq M \\ 0 < Tn \leq N \\ 0 < Tr \leq R \\ 0 < Tc \leq C \end{array}\right\} \tag{2}$$

where M = Number of output feature maps; N = Number of input feature maps; R = Row, C = Column, $<Tm, Tn> =$ Unroll factor. The generated hardware using the pseudocode 2 is shown in Fig. 2.

The data reuse principle is applied for input feature map and kernel weights are updated online. Because of the limited hardware resources on FPGA, $<Tm, Tn> = <16, 16>$ unroll factor is adopted. The computation engine is implemented in a tree-shaped poly structure and 16 inputs are coming from the input feature map and kernel weights. To unroll the loop Tm, 16 poly structures are duplicated. The total number of PEs in proposed accelerator design are 16. The processing element is shown in Fig. 3 and it can be observed that 16 inputs from input feature map and kernel weights are convolved to produce output. Therefore, a total of 16 multipliers and 15 adders are utilized to perform convolution operation in each processing element.

To implement hardware on FPGA, the performance of two different approaches of implementation can be 90% different from each other [18]. CNN accelerator design may require hundreds to thousands of hardware MAC units on FPGA. It creates a problem considering the computation cost

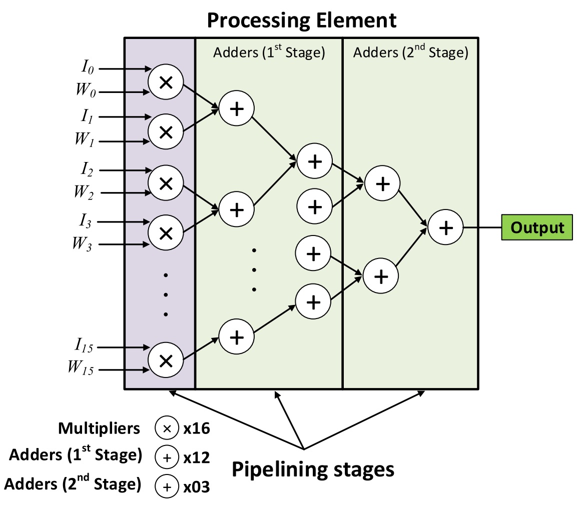

FIGURE 3. Hardware mapping of processing element for convolution layers.

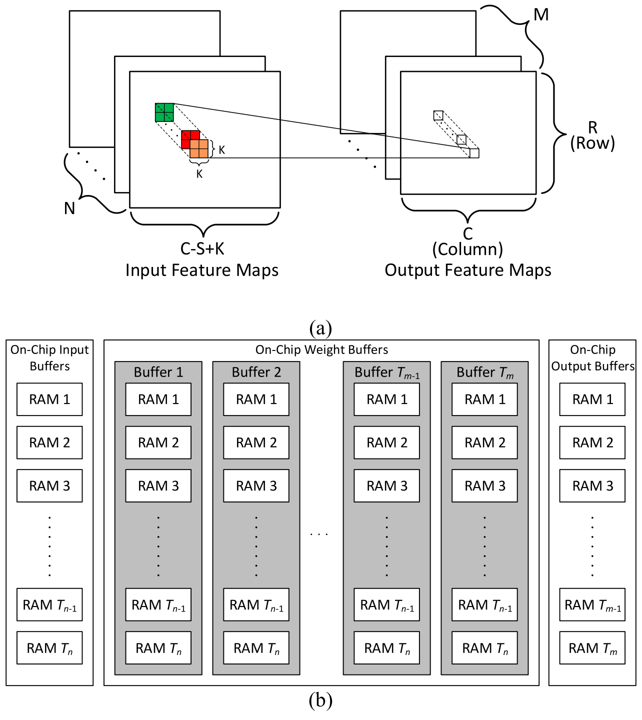

FIGURE 4. (a) Convolution graph (b) On-chip memory organization ($T_n = T_m = 16$).

on FPGA because MAC units are limited in number and also consume area and power [24].

Similarly, there is another bottleneck in convolution is the use of a deep binary adder tree to perform the addition. If we are going to implement the full hardware mapping of the convolution layer, it requires $N_cK_xK_y - 1$ binary adders on FPGA after the multiplication phase as represented in (1). The complexity of the adder tree can be realized such that only the first layer of AlexNet consumes 69% of resources [20]. Therefore, an optimization is required to replace the MAC units and adder tree structure. The main objectives are to reduce the amount of hardware resources

TABLE 1. Comparison of required number of PEs and on-chip data storage.

	Before	After
PEs Implemented Simultaneously	1	$T_n \times T_m$
Data Storage on on-chip cycle	$2 \times N \times K \times K$	$((T_r - 1)S + K)((T_c - 1)S + K)T_n + T_n \times T_m \times K \times K$

where $0 < T_r \leq R$ and $0 < T_c \leq C$

and make system energy efficient. In this design, multipliers are going to be replaced by a modified Booth encoding algorithm. Similarly, a typical binary adder tree is replaced by WALLACE tree-based adders.

C. MEMORY ORGANIZATION

The convolution graph is shown in Fig. 4 (a). The input feature maps N are convolved by a shifting window of $K \times K$ kernel weights to generate one pixel in an output feature map. The stride of the sliding window is S which is normally smaller than K. M output feature maps will form the set of input feature maps for the next layer. Therefore, memory structure is designed based on the sizes of input feature maps, output feature maps, and the number of kernel weights.

The convolution operation requires frequent switching of different input features and convolution kernels, which results in great data access pressure. To lower the cost of data access from external memory, increase data reuse, and to adjust the dataflow between the adjacent levels of memory hierarchy, loop unrolling, loop tiling and loop interchange are applied to customize the computation and communication patterns of the accelerator with three levels of memory hierarchy. As a result, the number of computational units that can be used simultaneously on an on-chip calculation is significantly increased, at the cost of only increasing the amount of data that is stored on-chip each time. Table 1 shows the comparison before and after the optimizations applied.

From (2) unroll factor is selected as $<Tm, Tn> = <16, 16>$. Therefore, after selecting this unroll factor, the PEs can be implemented simultaneously from 1 to $T_n \times T_m$ as shown in Table 1. The computation engine is implemented in a tree-shaped poly structure as discussed in the previous section and shown in Fig. 2. In Table 1, consider $W_{in} = ((T_r - 1)S + K)((T_c - 1)S + K)T_n$ and $W_{weight} = T_n \times T_m \times K \times K$ are the buffer sizes of memory accesses to input feature maps and kernel weights respectively. With specific tile size selection $<T_m, T_n, T_r, T_c>$, the number of memory accesses is changed from $2 \times N \times K \times K$ to $W_{in} + W_{weight}$.

In FPGA, block RAM (BRAM) comes as a single port or dual port. It means that one value can be accessed at a single clock cycle. However, it is not feasible for a design like CNN. In our proposed system, multi-dimensional memory mapping is designed to facilitate the access of multiple data simultaneously. Fig. 4 (b) shows the memory organization and it consists of input buffers, output buffers, and weight buffers. Each buffer set contains independent buffer banks. The number of input buffer banks is equal to T_n and

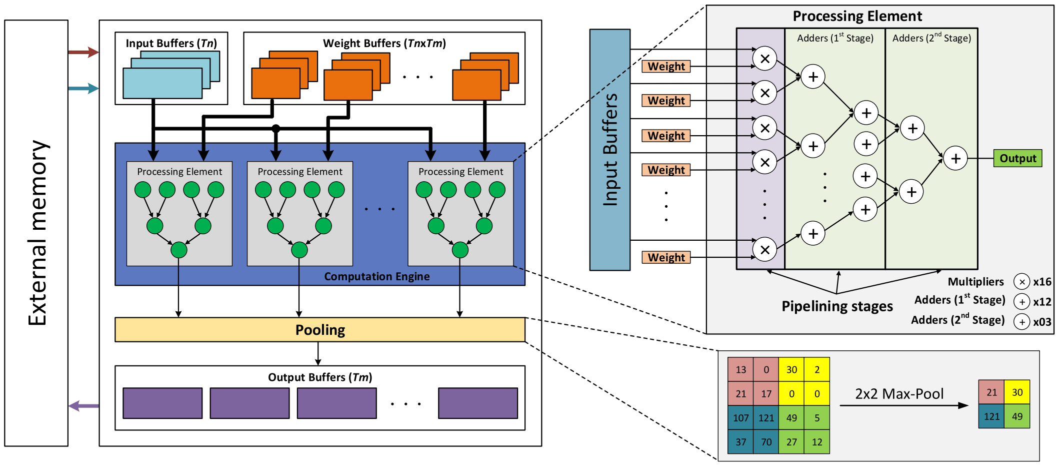

FIGURE 5. The parallel structure of proposed CNN accelerator design and a PE unit.

output buffer banks are equal to T_m as shown in Fig. 4 (b). The weight buffers contain multiple memories in a group and the data is arranged according to computation required to be performed as depicted in Fig. 2.

FPGA receives data from external DDR memory on the AXI interface. DMA is controlling the flow of data between external memory and on-chip memories. The commands from the processing system (PS) are issued to the programmable logic (PL) using the AXI-Lite interface and the internal controller is controlling the flow of data depending on the commands which are received. In order to exchange data effectively, Ping-Pong memory mode is applied for input and output buffers. The data flow and parallel structure of CNN accelerator design is represented in Fig. 5. It can be observed that PE is taking the parallel inputs from feature maps and kernel weights buffers to perform the convolution. Furthermore, three stages of pipelining are implemented in each PE unit to optimize the calculation order.

D. ACTIVATION FUNCTION AND POOLING LAYER

In CNN architecture before the pooling layer, activation function is applied to transform the input. The purpose of the activation function is to introduce non-linearity. Many activation functions are used in CNN architecture like Sigmoid, Tanh etc. However, in this accelerator design, Leaky ReLU is applied. The benefit of leaky ReLU is that it can help to prevent the neurons from dying during training.

The pooling layer is employed to reduce the dimensions of feature maps. It helps to reduce the computations in the network and also discard the unnecessary information. Max-pooling and average-pooling are typical pooling functions. In this proposed design, max-pooling is used as shown in Fig. 5.

E. BATCH NORMALIZATION

It is known that batch normalization offers many different improvements. For example, it can help in speeding up the training time. It has been observed that the batch normalization improves the accuracy of the network over that of the one without using the batch normalization, for typical YOLO networks. However, it is not necessary to use the batch normalization for a small network like tiny-YOLO-v2 to speed-up the feedforward path [25]. Therefore, in the proposed accelerator design, batch normalization is not implemented.

F. PRIOR WORKS ON MULTIPLIER DESIGN

Convolution is multiplication or dot-product of input feature maps and kernel weights. To implement the convolution, MAC is necessary to perform multiplication. However, the MAC unit has a problem with consuming area and power. It contains a multiplier and accumulator, and each multiplier on FPGA or ASIC costs a large number of logic gates and high power [26]. To implement parallel multipliers, it consists of three operations such as partial product generation, partial product reduction, and final addition by carry propagate adder (CPA) [27].

To multiply two L bit numbers a and b, the partial products can be generated either by using an ANDing operation or by implementing a modified Booth encoding algorithm [28]. The first method generates partial product PP_L by ANDing each bit b_L of the multiplier with all the bits of the multiplicand a. Fig. 6 shows the generation of partial products using ANDing operation.

After the generation of partial products, there is a need to reduce the PPs efficiently. Considering a $L_1 \times L_2$ multiplier, different techniques are used to reduce L_1 layers of the partial products to two layers for their final addition using any CPA.

WALLACE reduction is one of the most commonly adopted schemes to reduce layers of partial products.

1) WALLACE TREE MULTIPLIER

In previous work [21], the WALLACE tree multiplier (WTM) was proposed to replace the MAC unit as shown in Fig. 7. In this design, the input feature map and kernel weights are coming as an input to

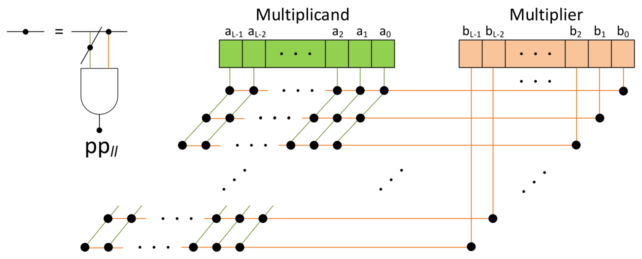
Multiplicand
a_{L-1} a_{L-2} · · · a_2 a_1 a_0
Multiplier
b_{L-1} b_{L-2} · · · b_2 b_1 b_0
=
pp_{ll}

FIGURE 6. Generation of partial products using ANDing operation.

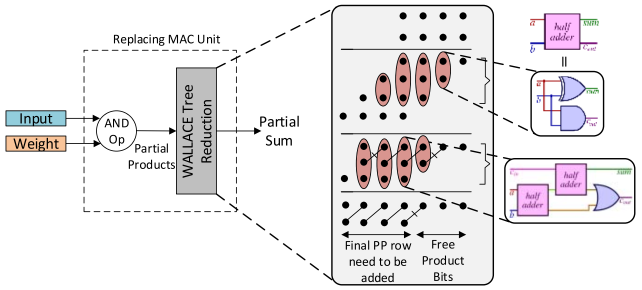

FIGURE 7. WALLACE tree multiplier (WM) for CNN accelerator.

TABLE 2. Radix-4 modified booth encoding and partial-product selection ($j = 2\alpha$).

b_{j+1}	b_j	b_{j-1}	b'_α	P_α	Comments
0	0	0	0	0	string of "0"s
0	0	1	1	$+A$	end of "1"s
0	1	0	1	$+A$	a single "1"
0	1	1	2	$+2A$	end of "1"s
1	0	0	–2	$-2A$	beginning of "1"s
1	0	1	–1	$-A$	a single "0"
1	1	0	–1	$-A$	beginning of "1"s
1	1	1	0	0	string of "1"s

TABLE 3. Radix-4 modified booth encoding partial-product generation.

P_α	$p_{\alpha,k}$	$p_{\alpha,k-1}$	$p_{\alpha,k-2}$	…	$p_{\alpha,2}$	$p_{\alpha,1}$	$p_{\alpha,0}$	E_α
0	0	0	0	…	0	0	0	0
$+A$	a_{k-1}	a_{k-1}	a_{k-2}	…	a_2	a_1	a_0	0
$+2A$	a_{k-1}	a_{k-2}	a_{k-3}	…	a_1	a_0	0	0
$-A$	$\overline{a_{k-1}}$	$\overline{a_{k-1}}$	$\overline{a_{k-2}}$	…	$\overline{a_2}$	$\overline{a_1}$	$\overline{a_0}$	1
$-2A$	$\overline{a_{k-1}}$	$\overline{a_{k-2}}$	$\overline{a_{k-3}}$	…	$\overline{a_1}$	$\overline{a_0}$	1	1

this multiplier. To generate the PPs, ANDing operation is performed as shown in Fig. 6. After generating the partial products, the WALLACE tree reduction is applied to produce the final product. To further simplify the design, combinational logic is used to implement the full adders (FA) and half adder (HA) which are based on the CSA technique depicted in Fig. 7.

For a $L \times L$ multiplier, L numbers of PPs are generated. For example, 16 number of partial products are generated for a 16-bit multiplier. Therefore, it consumes an effective hardware and there are six FA adder delays in its path [21]. Reducing the number of PPs is another optimization technique. There is a need to replace the WALLACE tree multiplier with a multiplier that generates less number of PPs. MBE is such a multiplier design that further reduces the number of PPs and as a result, consumes fewer hardware resources compared to the WALLACE tree multiplier. Therefore, the modified Booth encoding multiplier is proposed and implemented to replace the WALLACE tree multiplier.

2) MODIFIED BOOTH ENCODING MULTIPLIER (RADIX-4)

Consider A and B are k-bit and l-bit two's complement integers respectively:

$$A = -a_{k-1} \cdot 2^{k-1} + \sum_{i=0}^{k-2} a_i \cdot 2^i \tag{3}$$

$$B = -b_{l-1} \cdot 2^{l-1} + \sum_{j=0}^{l-2} b_j \cdot 2^j \tag{4}$$

Interestingly, the Booth encoding algorithm can be used for two's complement as well as unsigned multipliers [29]. It is to note that firstly "0" value is set for b_{-1} and appended to the rightmost of B. Consider a multiplier for two's complement, l should be even otherwise B is one-bit sign-extended to ensure l is even. Similarly for unsigned, B is zero-extended with one "0" for odd values of l to make it even and if it is already even, B is zero-extended with two "0"s.

Since it is Booth encoded algorithm and B is encoded from a group of three bits to two bits. For each $j \in \{0, 2, 4, \ldots, l-2\}$, b_{j+1}, b_j and b_{j-1} are encoded to b'_α which is a signed digit. α considered here is $\alpha = j/2$ and $b'_\alpha = -2b_{j+1} + b_j + b_{j-1}$. Each PP is calculated by multiplication of multiplier A and b'_α. The final product is computed as follows:

$$P = \sum_{\alpha=0}^{l/2-1} P_\alpha \cdot 2^{2\alpha} = \sum_{\alpha=0}^{l/2-1} A \cdot b'_\alpha \cdot 2^{2\alpha} \tag{5}$$

Final Booth encoding and PP selection are summarized in [30, Tab. 2].

The multiplicand A is copied if the partial product is $+A$ after doing encoding. Since each PP is generated using a pair of 2 bits of the multiplier, the PP is shifted by two places to the left. For a PP to be $+2A$, there will be a left shift of one bit for multiplicand before selection. For the encoded digit of the multiplier that is $-A$, the two's complement of the multiplicand is copied. For the last encoded of $-2A$, the two's complement of the PP is further shifted left by one-bit position. PP generation is shown in Table 3 for each selection. It is noted that there will be $k+1$ bits in the PP to compensate the left shift of A and if A is not shifted then the extra bit will be sign extension. The value "0" is set for correction bit E_α to perform addition or value "1" for compensation of two's complement. It is placed in the LSB position of the PP.

Each PP is sign-extended and there is a need to eliminate the sign extension logic. Sign extension logic is performed by inverting the sign bit, a value "1" is added to the same

TABLE 4. Radix-4 modified booth partial-product matrix, $k = l = 6$.

Column	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
						1						
$P_0 \cdot 2^0$:					1	$\overline{p_{0,6}}$	$p_{0,5}$	$p_{0,4}$	$p_{0,3}$	$p_{0,2}$	$p_{0,1}$	$p_{0,0}$
$P_1 \cdot 2^2$:			1	$\overline{p_{1,6}}$	$p_{1,5}$	$p_{1,4}$	$p_{1,3}$	$p_{1,2}$	$p_{1,1}$	$p_{1,0}$		E_0
$P_2 \cdot 2^4$:	1	$\overline{p_{2,6}}$	$p_{2,5}$	$p_{2,4}$	$p_{2,3}$	$p_{2,2}$	$p_{2,1}$	$p_{2,0}$		E_1		
								E_2				
P	p_{11}	p_{10}	p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

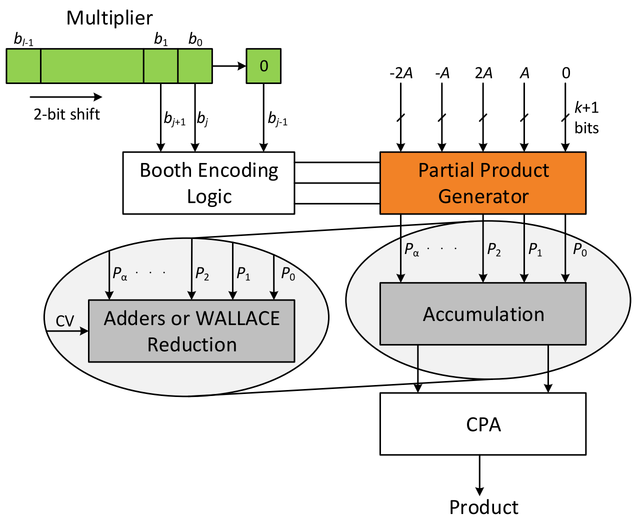

FIGURE 8. The MBE Multiplier.

column, and PP is extended by constant "1"s. Now to eliminate the sign extension logic, all these "1"s are added offline to form a correction vector (CV) [28]. The number of "1"s can be reduced by pre-adding the constants [30]. The simplified PP matrix is shown in Table 4 considering a 6×6 multiplier [31], [32]. In Table 4, the conventional modified booth encoding algorithm generates $L/2 + 1$ number of PP rows rather than $L/2$ because of extra correction bits which are placed at the least significant bit position of each partial product row for negative encoding as shown in Table 3.

III. PROCESSING ELEMENT BASED ON MBE MULTIPLIER AND WALLACE TREE ADDERS

A new processing element based on MBE multiplier to replace MAC unit and WALLACE tree adders as an alternate solution for deep binary adder tree is proposed. The hardware cost is reduced and high power efficiency is achieved with the proposed methods.

A. MBE DESIGN FOR MULTIPLICATION

It is discussed that multiplication consists of three major steps: 1) encoding and generating partial products; 2) reducing the partial products by reduction schemes (e.g., Wallace tree [33], [34]) to final two rows; and 3) adding the remaining two rows of partial products by using a carry propagate adder to obtain the final product. In this proposed design, the focus is on the first two-step to reduce hardware cost, delay, and power consumption of proposed multiplier [35]. The MBE technique is widely applied in parallel multipliers because it can reduce the number of PP rows to half and thus reducing the size and enhancing the speed of reduction tree [35]–[37]. The MBE multiplier block diagram is shown in Fig. 8. Without error correction bits, the number of PPs with sign extension logic will be $L/2$ and a simple accumulation process can be used to generate the final product. However, this technique costs more hardware resources and power. The sign extension elimination logic is implemented in our proposed design as described in Table 3. However, the sign extension elimination generates one extra PP and number of PPs changes from $L/2$ to $L/2+1$ due to error correction bits (as a CV) due to negation. Applying the WALLACE tree reduction scheme after the generation of PPs with MBE can further improve the multiplier delay [38]. The final product is generated after the CPA.

TABLE 5. Height of WALLACE tree using (4:2) with parallel inputs.

Number of Inputs	Height of Tree
4	1
$5 \le n \le 8$	2
$9 \le n \le 16$	3
$17 \le n \le 32$	4

Three MBE multiplier designs are implemented in this work to replace the MAC units in CNN accelerator design as shown in Fig. 9. Fig. 9 (a) shows the MBE multiplier with sign extension is called hereafter as BMS and Fig. 9 (b) shows the MBE multiplier with sign extension elimination technique using a correction vector (BMSE). Fig. 9 (c) shows the MBE multiplier and WALLACE tree for PPs reduction along with the sign extension elimination technique are named as BMSEW. Therefore, an efficient BMSE multiplier is selected for proposed PE unit considering power and hardware consumption. It also consumes fewer logic gates because no sign extension hardware is implemented in this multiplier design.

B. ADDERS BASED ON WALLACE TREE REDUCTION

WALLACE tree is mostly used in multiplier architectures. The objective is to reduce the layers of PPs generated during multiplication. Carry save addition is applied in WALLACE to avoid the carry propagation delay in its path. The PPs are

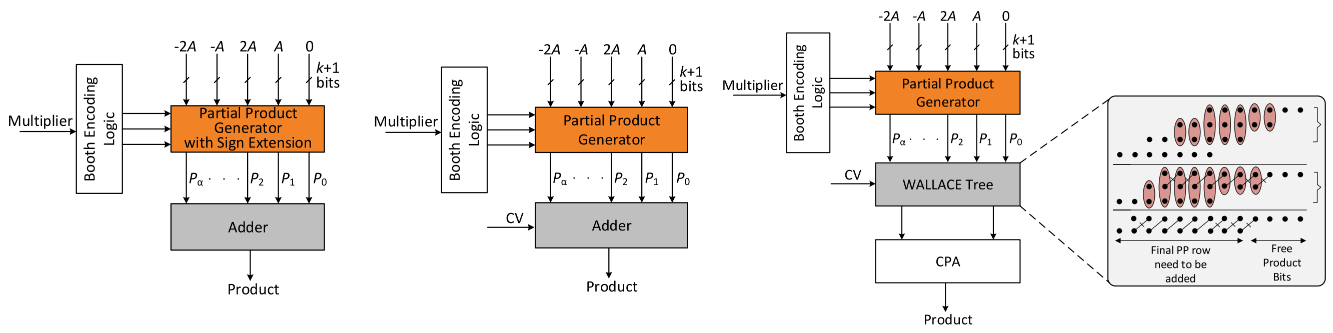

FIGURE 9. Multiplier designs replacing MAC unit (a) MBE multiplier with sign extension (BMS) (b) MBE multiplier with sign extension elimination (BMSE) (c) MBE multiplier along with WALLACE reduction for PPs with sign extension elimination (BMSEW).

TABLE 6. Tiny-YOLO-v2 architecture.

Layer	Input	Filters	Size	S*	Output	Precision loss (%)
Conv1	224×224×3	16	3×3	1	224×224×16	0.0
MP*	224×224×16		2×2	2	112×112×16	
Conv2	112×112×16	32	3×3	1	112×112×32	0.2
MP	112×112×32		2×2	2	56×56×32	
Conv3	56×56×32	64	3×3	1	56×56×64	0.1
MP	56×56×64		2×2	2	28×28×64	
Conv4	28×28×64	128	3×3	1	28×28×128	0.1
MP	28×28×128		2×2	2	14×14×128	
Conv5	14×14×128	256	3×3	1	14×14×256	0.1
MP	14×14×256		2×2	2	7×7×256	
Conv6	7×7×256	512	3×3	1	7×7×512	0.1
MP	7×7×512		2×2	1	7×7×512	
Conv7	7×7×512	1024	3×3	1	7×7×1024	0.1
Conv8	7×7×1024	1024	3×3	1	7×7×1024	0.0
Conv9	7×7×1024	125	1×1	1	7×7×125	0.0

Note (*): MP = Max pooling; S = Stride

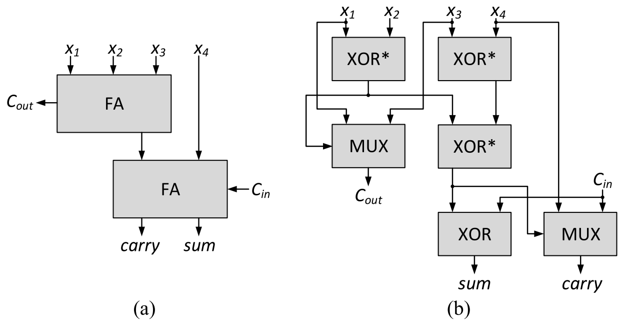

FIGURE 10. (a) Structure of (4:2) compressor. (b) Optimized implementation of exact (4:2) compressor.

first reduced to two numbers using CSA tree then these two numbers are added by CPA to get the final product. A reduction technique based on a (3:2) compressor using full adder reduces three layers of PPs to two and (2:2) compressor reduces two layers to two using half adder. WALLACE tree-based adders using (3:2) and (2:2) compressors were used to replace the typical binary adder tree [22]. As a result, it reduces LUTs consumption and improves the propagation delay. Although the (3:2) compressor is well known but in research, many higher-order compressor designs have been proposed to further reduce the area, power, and delay characteristics [39].

In this article, (4:2) compressor is selected to further improve the system performance compared to [22]. The internal structure of the exact (4:2) compressor is shown in Fig. 10 (a) and it consisted of two full adders connected serially. The three outputs of this design are given below:

$$Cout = (x1 \oplus x2)x3 + \overline{(x1 \oplus x2)}x1 \tag{6}$$

$$sum = x1 \oplus x2 \oplus x3 \oplus x4 \oplus Cin \tag{7}$$

$$carry = (x1 \oplus x2 \oplus x3 \oplus x4)Cin + \overline{(x1 \oplus x2 \oplus x3 \oplus x4)}x4 \tag{8}$$

The optimized implementation of the exact (4:2) compressor design using XOR-XNOR gates is shown in Fig. 10 (b) [40]. It is based on three XOR-XNOR gates (represented as XOR*), one XOR and two 2-1 MUX. Consider the unitary delay of Δ by any gate then (4:2) compressor has a delay of 3Δ giving the delay of 1Δ less compared to the conventional implementation [40]. Applying this structure into WALLACE tree adders, each level of a tree with (4:2) compressor reduces the number of operands by a factor of 2. This reduction continues until we get the final two rows. The height of a tree can be represented as:

$$h_{(4:2)}(n) = \left\lceil \log_2(n/2) \right\rceil \tag{9}$$

where $h_{(4:2)}(n)$ is the height of an adder tree for n-number of parallel inputs after multiplication. The height of this reduction tree of WALLACE based on (4:2) compressor for parallel inputs are shown in Table 5. If we compare the two compressor designs in terms of LUTs consumption, the WALLACE adders based on (4:2) compressor in the first stage of adder tree in PE unit consumes only 176 LUTs as compared to 188 LUTs based on (3:2) compressor in [22].

A new PE unit based on MBE multiplier and WALLACE tree-based adders using (4:2) compressor is presented in Fig. 11. Previously, the WALLACE tree multiplier was used to perform the convolution of input feature maps and convolution kernels [21]. In this work, the MBE multiplier is proposed and implemented to replace the WALLACE tree

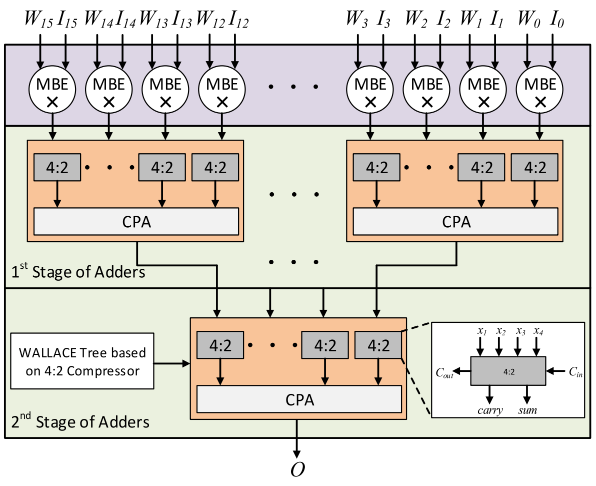
W_{15} I_{15} W_{14} I_{14} W_{13} I_{13} W_{12} I_{12}
W_3 I_3 W_2 I_2 W_1 I_1 W_0 I_0
MBE ×
4:2
CPA
1st Stage of Adders
WALLACE Tree based on 4:2 Compressor
2nd Stage of Adders
x_1 x_2 x_3 x_4
C_{out}
C_{in}
carry
sum
O

FIGURE 11. The final proposed design for the PE unit that will replace the old PE units in CNN accelerator design.

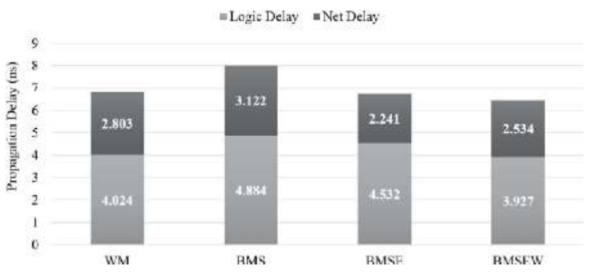

FIGURE 12. Propagation delay of different multiplier designs.

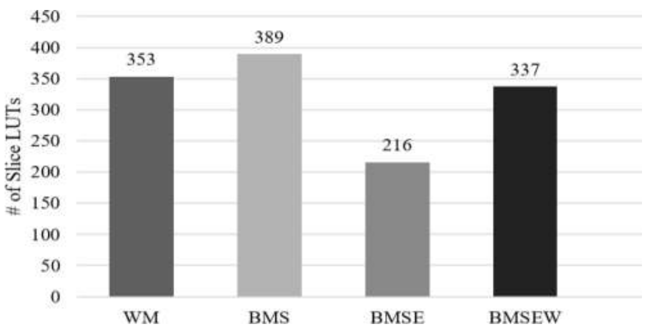

FIGURE 13. Device utilization summary of multiplier designs.

TABLE 7. Implementation results of proposed design and previous PE units.

PE unit for convolution	# LUTs	# Mult.	# DSPs	# Adders	Adder Type
WM [14]	6149	16	0	15	Built-in adders
WM+WAT [21, 22]	5943	16	0	05	Wallace tree adders with 3:2 compressor
Conv. block [42]	5124 (*)	18	18	17	Built-in adders
BMSE+WAT (Our work)	3612	16	0	05	Wallace tree adders with 4:2 compressor

Note (*): LE: Logic elements

TABLE 8. Power and frequency of PE unit for different multipliers.

PE unit design	Power (mW)	Freq. (MHz)	Mult. Type	Adder Type
WM in [14]	293	177	WALLACE	Built-in Adders
WM+WAT [21, 22]	271	522	WALLACE	Wallace tree adders with 3:2 compressor
MAC in [14]	226	177	DSP48E1 Slice	Built-in Adders
BMSE+WAT (Our work)	176	533	BMSE	Wallace tree adders with 4:2 compressor

multiplier due to hardware overhead, power consumption, and a long delay in its path. It will also avoid the usage of bulky MACs. As a result of multiplication, 16 outputs are generated. These parallel outputs become the inputs of WALLACE tree adders (WAT). Each PE unit produces a partial sum after the process of multiplication and addition performed by WAT.

In the next section, the discussion and results are presented about the overall system performance with the previous approaches.

IV. EXPERIMENTAL RESULTS AND DISCUSSION

In this article, Xilinx Zynq706 (XC7Z045 SoC) board is used for the evaluation of a proposed design and Verilog HDL is adopted to implement the architecture. Behavioral simulation is performed with ModelSim SE 10.5, synthesis and implementation is done with Vivado 2015.4. The proposed design is based on pure RTL design flow and it does not use the HLS flow. As a result, hardware optimizations are performed by implementing our soft IP design using RTL written in Verilog HDL.

A. NETWORK ARCHITECTURE

In order to test the performance and efficiency of the proposed architecture in deeper networks, Tiny-YOLO-v2 architecture is used. It consists of 9 convolution layers and 6 max-pooling layers as shown in Table 6. The model is trained using a PASCAL VOC 2012 dataset [41]. It consisted of 20 classes and approximately 5000 training images in its dataset. The network is trained in a 32-bit environment on GPU and stored in a floating-point but an inference phase; this high precision is not required. Therefore, 16-bit fixed-point number representation is selected considering the precision loss from PASCAL VOC pre-trained weights from the Darknet framework. Table 6 shows the precision loss costed by using a 16-bit fixed-point for each convolution layer. The proposed system architecture on FPGA is using 16-bit fixed-point quantization and it incurs an accuracy loss of approximately 1.9% compared with the full precision network. It also gives the mean average precision (mAP) of 51.12%.

B. IMPLEMENTATION RESULTS AND DISCUSSION

1) DELAY AND HARDWARE RESOURCES OF MULTIPLIER DESIGNS

Four multiplier designs are synthesized using Vivado 2015.4 and the results are shown in Fig. 12 and Fig. 13 for path

TABLE 9. Comparison of the proposed design with the previous implementations.

	FPGA'15 [18]	Tincy YOLO [43]	OpenCL Tiny-Yolo-v2 [44]	Tiny-Yolo-v2 [14]	Lightweight YOLO-v2 [45]	Tiny-Yolo-v2 [46]	Our Work (Tiny-Yolo-v2)
Precision	32-bits float	(1-b W*, 3-b A*)	16-bits fixed	16-bits fixed	(1-b W*, 32-b A*)	(1-b W*, 6-b A*)	**16-bits fixed**
Frequency	100 MHz	-	117 MHz	120 MHz	300 MHz	200 MHz	**180 MHz**
FPGA Chip (Process)	Virtex7 VX485T (28 nm)	Zynq Ultrascale+ XCZU3EG (16 nm)	Cyclone V (28 nm)	Cyclone IV (60 nm)	Zynq Ultrascale+ XCZU9EG (16 nm)	Virtex7 VX485T (28 nm)	**Zynq706 XC7Z045 (28 nm)**
LUT Type	6-input LUT	6-input LUT	8-input LUT	4-input LUT	6-input LUT	6-input LUT	**6-input LUT**
LUTs	186.3K	-	72.7K (†)	28K (††)	135K	86K	**64.9K**
DSPs	2240	-	122	256	377	168	**0**
CNN Size (GOP)	1.33	4.5	6.97	2.02	14.97	6.97	**2.02**
Image Size	224×224	416×416	416×416	224×224	224×224	416×416	**224×224**
Frame Rate	46.33	16	3.8	29.47	40.81	66.56	**43.08**
Accuracy (mAP) %	-	48.5	-	-	67.6	51.38	**51.12**
Power Efficiency (GOP/s/W)	3.31	12	-	44.1	-	53.29	**61.64**
Power Efficiency w.r.t. FPGA Process (GOP/s/W×nm)	93	192	-	2646	-	1492	**1726**

Note (*): W: Weight; A: Activation
Note (†): ALMs: Adaptive logic modules
Note (††): LEs: Logic elements

TABLE 10. Number of MACs required for different designs.

	[18]	[47]	[14]	Our work
Unroll factor <*Tm*,*Tn*>	<64,7>	<64,6>	<16,16>	<16,16>
MACs in one PE Unit	35	30	16	0
Total # MACs	2240	1926	256	0

delay and device utilization on FPGA respectively. In Fig. 12, the path delay faced by each multiplier is further divided into its logic and net delay. BMS multiplier design faces the highest delay due to the sign extension logic used in this multiplier. Consequently, this multiplier utilizes more adders to generate the final product and hence it faces more delay. On the other hand, the path delay of BMSEW is lowest amongst the other multipliers because WALLACE reduction using in MBE helps to reduce the path delay [38]. Its delay is reduced by 19% while comparing with the MBE multiplier design (BMS). This BMSEW design faces only four adders delay in its critical path compared to six FA delay of WALLACE tree multiplier for 16-bit operands [21]. The path delay for WALLACE based architectures depends on the number of PPs and it can be calculated and verified using O(log3/2(N)) [21], [38].

Similarly, Fig. 13 shows the device utilization summary for different multiplier designs. It can be observed that BMSE multiplier design consumes fewer hardware resources while comparing to others. On the other hand, MBE with WALLACE reduction and sign extension elimination design (BMSEW) consumes relatively high LUTs as compared to MBE without WALLACE (BMSE) because many adders are used for parallel processing. Comparing the delay and hardware costs of MBE multipliers with WM, it can be observed from Fig. 12 that the timing of BMSE is improved by 0.8% and 5.4% of BMSEW comparing with WM. Similarly, Fig. 13 shows that the BMSE LUTs consumption is improved by 38.8% and 4.5% of BMSEW compared to WM. Therefore, multiplier design can be selected depending on the requirement of the criticality of hardware resources or considering the delay-sensitive designs. The main focus of this article is to save the amount of hardware resources, therefore BMSE multiplier is used to perform the multiplication task in convolution.

2) POWER, FREQUENCY, AND RESOURCES OF PROPOSED PE UNIT

From the above discussion and results, the multiplier design of BMSE is proposed in the PE unit as shown in Fig. 11. Further, to perform the addition on intermediate results generated in a process of multiplications, WALLACE adders based on (4:2) compressor are implemented as shown in Fig. 11. Table 7 shows the synthesis results of one PE unit of proposed design on FPGA and different architecture designs presented in the literature. The listed works in Table 7 are not based on the same FPGAs and the resources and DSP architecture may differ from each other. However, still some aspects of different designs can be compared, such as the hardware overhead, since the architectures of the mainstream FPGAs are quite similar. In [14], the PE unit is regenerated and the MAC unit is replaced with WM. It has utilized 16 multipliers and 15 numbers of adders to perform convolution in one PE unit. Similarly, a PE unit is regenerated using the multiplier design of [21] and addition is performed with WALLACE tree adders [22]. This PE design used 05 blocks of 16x4 WALLACE tree adders to perform addition on intermediate results after multiplication. The design of [42] is based on convolution blocks that perform the convolution task using multipliers and adders. A total of 18 multipliers and 17 adders are used to perform the convolution task. Our proposed design of the PE unit based on the BMSE multiplier with 05 blocks of WAT using a (4:2) compressor achieves high hardware efficiency as compared to previous designs. The LUTs consumption of the proposed PE unit is improved by 41.2% compared to [14] and 29.5% to [42]. Therefore, this PE unit design of Fig. 11 is proposed and applied in our final architecture.

The next important parameter of efficient hardware is power efficiency. In this proposed design using MBE multiplier, the focus is to reduce the area, delay, and power consumption of multipliers [35]. Therefore, from Table 7 it can be observed that the hardware costs of PE unit are significantly reduced. Similarly, power is measured for different PEs and proposed design. Table 8 shows the power consumption and synthesis frequency of PE unit designs with different multipliers. It can be observed that the power of PE using WALLACE multiplier consumes more power while comparing with WALLACE multiplier with WALLACE adders. It means adder tree is another factor that consumes hardware resources and power. It is discussed in Section III that replacing the built-in adders with WAT can improve the hardware consumption, delay, and power. However, DSP based PE unit design still achieves low power compared to the other two PEs with WALLACE multiplier designs. The power and frequency of proposed PE unit design of Fig. 11 is compared with other designs. The power consumption of the proposed PE is 22.1% of that of the MAC-based PE unit. In the meanwhile, the maximum achievable frequency is also improved by 201% when the design is changed from built-in adders to WALLACE tree adders because built-in adders are slow and consume more hardware resources.

The system architecture is built on the PE unit as depicted in Fig. 11 and Table 9 represents the comparison of proposed architecture with the previous designs. Only FPGA'15 [18] design is running the AlexNet model and the rest of the designs in Table 9 are running the Tiny-YOLO-v2 model on FPGA platform to test their proposed designs. The bulky MAC units (DSP48 slices in FPGA) are not used and further typical deep binary adder tree is replaced. In this work, custom BMSE multiplier is implemented instead of DSP48 slices and WALLACE based adders with (4:2) compressor is to perform addition. Note that the achieved peak performance is 89.28 GOP/s and the actual throughput is about 87.03 GOP/s with the extra clock cycles for memory access when the proposed CNN is used for an object detection system. It can be observed that the proposed system reduces hardware cost by 24.5% while attaining a power efficiency of 61.64 GOP/s/W compared to the previous work for the object detection task [46]. Similarly, it also gives us an improvement in a frame rate of 13.61 and power efficiency by 17.54 GOP/s/W compared with the design of [14]. Considering all previous designs, zero number of DSPs are used in our work. Therefore, the power efficiency of 61.64 GOP/s/W is achieved with fewer hardware resources as compared to previous designs. The proposed FPGA based design is free from the DSP48 slices.

Since this design is free from the bulky MAC units, it can be a good candidate for power and hardware efficient systems. Table 10 shows previous designs with different unroll factors. It is also presented that how many numbers of MACs are required to perform the convolution task with different unroll factors. MACs are not recommended and require an alternate solution due to high area and power consumption [26]. The proposed design in this article is free from the MAC units instead implemented a custom multiplier. Table 10 shows that zero number of MACs are consumed.

V. CONCLUSION

In this paper, optimization techniques are proposed and implemented to achieve better system performance, reduced amount of hardware resources, and power efficiency of the CNN accelerator design for object detection. MBE multiplier is presented to replace the MAC units on FPGA. Another challenge was to find an alternate solution for a typical adder tree. Therefore, WALLACE tree-based adders with (4:2) compressors are used to replace the traditional adder tree. The objective was to reduce the resources on FPGA consumed by CNN accelerator and improve the system performance in terms of power. More computations can be implemented on available hardware because proposed design helps to reduce hardware consumption compared to previous designs. Achieving high power efficiency can help us to use the system in practical low power applications.

The proposed system architecture and optimization techniques opens new opportunities for ASIC implementation targeting the low area and power applications. To test the behavior of proposed architecture, Xilinx FPGA platform is selected to implement the Tiny-YOLO-v2 network for object detection. It validates that the proposed hardware optimization solutions will help us to achieve high performance in terms of resource utilization and power for future system implementations.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2012, pp. 1106–1114.

[2] R. Sivaramakrishnan, C. Sema, K. Incheol, T. George, and A. Sameer, "Visualization and interpretation of convolutional neural network predictions in detecting pneumonia in pediatric chest radiographs," *Appl. Sci.*, vol. 8, no. 10, p. 1715, 2018.

[3] L. Yinghua, S. Bin, K. Xu, D. Xiaojiang, and G. Mohsen, "Vehicle-type detection based on compressed sensing and deep learning in vehicular networks," *Sensors*, vol. 18, p. 4500, Dec. 2018.

[4] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Boston, MA, USA, 2015, pp. 1–9, doi: 10.1109/CVPR.2015.7298594.

[5] S. Ren, K. M. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal network," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.

[6] O. Abdel-Hamid, A. R. Mohamed, H. Jiang, and G. Penn, "Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, Kyoto, Japan, Mar. 2012, pp. 4277–4280.

[7] C. Farabet, C. Poulet, J. Y. Han, and C. Y. Le, "CNP: An FPGA-based processor for convolutional networks," in *Proc. Int. Conf. Field Program. Logic Appl.*, Prague, Czech Republic, Aug./Sep. 2009, pp. 32–37.

[8] M. Sankaradas *et al.*, "A massively parallel coprocessor for convolutional neural networks," in *Proc. IEEE Int. Conf. Appl. Syst. Architect. Process.*, New York, NY, USA, Jul. 2009, pp. 53–60.

[9] R. Hadsell *et al.*, "Learning long-range vision for autonomous off-road driving," *J. Field Robot.*, vol. 26, no. 2, pp. 120–144, 2009, doi: 10.1002/rob.20276,

[10] J. Maria, J. Amaro, G. Falcao, and L. A. Alexandre, "Stacked autoencoders using low-power accelerated architectures for object recognition in autonomous systems," *Neural Process. Lett.* vol. 43, pp. 445–458, May 2016.

[11] J. Cheng, P.-S. Wang, G. Li, Q.-H. Hu, and H.-Q. Lu, "Recent advances in efficient computation of deep convolutional neural networks," *Front. Inf. Technol. Electron. Eng.*, vol. 19, no. 1, pp. 64–77, Jan. 2018.

[12] D. Aysegul *et al.*, "Accelerating deep neural networks on mobile processor with embedded programmable logic," in *Proc. IEEE NIPS*, 2013, p. 1.

[13] M. Peemen, A. A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," in *Proc. IEEE 31st Int. Conf. Comput. Design (ICCD)*, 2013, pp. 13–19.

[14] W. Xie, C. Zhang, Y. Zhang, C. Hu, H. Jiang, and Z. Wang, "An energy-efficient FPGA-based embedded system for CNN application," in *Proc. IEEE Int. Conf. Electron Devices Solid-State Circuits (EDSSC)*, 2018, pp. 1–2, doi: 10.1109/EDSSC.2018.8487057.

[15] E. Nurvitadhi *et al.*, "Can FPGAs beat GPUs in accelerating next-generation deep neural networks?" in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays (FPGA)*, 2017, pp. 5–14.

[16] Y. Ma, N. Suda, Y. Cao, S. Vrudhula, and J.-S. Seo, "ALAMO: FPGA acceleration of deep learning algorithms with a modularized RTL compiler," *Integr. VLSI J.*, vol. 62, pp. 14–23, Jun. 2018. [Online]. Available: doi.org/10.1016/j.vlsi.2017.12.009

[17] J. Qiu *et al.*, "Going deeper with embedded FPGA platform for convolutional neural network," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays (FPGA)*, 2016, pp. 26–35.

[18] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays (FPGA)*, 2015, pp. 161–170.

[19] K. Abdelouahab, M. Pelcat, J. Serot, C. Bourrasset, and F. Berry, "Tactics to directly map CNN graphs on embedded FPGAs," *IEEE Embedded Syst. Lett.*, vol. 9, no. 4, pp. 113–116, Dec. 2017.

[20] K. Abdelouahab, M. Pelcat, and F. Berry, "The challenge of multi-operand adders in CNNs on FPGAs: How not to solve it!" in *Proc. 18th ACM Int. Conf. Embedded Comput. Syst. Architect. Model. Simulat. (SAMOS)*, Jul. 2018, pp. 157–160.

[21] F. U. D. Farrukh, T. Xie, C. Zhang, and Z. Wang, "Optimization for efficient hardware implementation of CNN on FPGA," in *Proc. IEEE Int. Conf. Integr. Circuits Technol. Appl. (ICTA)*, Beijing, China, 2018, pp. 88–89, doi: 10.1109/CICTA.2018.8706067.

[22] F. U. D. Farrukh, T. Xie, C. Zhang, and Z. Wang, "A solution to optimize multi-operand adders in CNN architecture on FPGA," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Sapporo, Japan, 2019, pp. 1–4, doi: 10.1109/ISCAS.2019.8702777.

[23] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *Artificial Neural Networks and Machine Learning—ICANN*. Cham, Switzerland: Springer, 2014, pp. 281–290.

[24] S. Sabeetha, J. Ajayan, S. Shriram, K. Vivek, and V. Rajesh, "A study of performance comparison of digital multipliers using 22nm strained silicon technology," in *Proc. 2nd Int. Conf. Electron. Commun. Syst. (ICECS)*, 2015, pp. 180–184.

[25] R. Huang, J. Pedoeem, and C. Chen, "YOLO-LITE: A real-time object detection algorithm optimized for non-GPU computers," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Seattle, WA, USA, 2018, pp. 2503–2510.

[26] J. Garland and D. Gregg, "Low complexity multiply-accumulate units for convolutional neural networks with weight-sharing," *ACM Trans. Architect. Code Optim.*, vol. 15, no. 3, pp. 1–24, Aug. 2018.

[27] J. E. Stine, "Multiplication," in *Digital Computer Arithmetic Datapath Design Using Verilog HDL*. New York, NY, USA: Springer, 2004, pp. 55–93, ch. 4.

[28] S. A. Khan, "Design options for basic building blocks," in *Digital Design of Signal Processing Systems: A Practical Approach*. New York, NY, USA: Wiley, 2011, pp. 183–252, ch. 5.

[29] O. L. MacSorley, "High-speed arithmetic in Binary computers," *Proc. IRE*, vol. 49, no. 1, pp. 67–91, Jan. 1961.

[30] E. G. Walters, "Array multipliers for high throughput in Xilinx FPGAs with 6-input LUTs," *Computers*, vol. 5, no. 4, pp. 1–25, 2016.

[31] M. D. Ercegovac and T. Lang, "Multiplication," in *Digital Arithmetic* (The Morgan Kaufmann Series in Computer Architecture and Design). San Francisco, CA, USA: Morgan Kaufmann, 2004, pp. 180–245, ch. 4.

[32] B. Parhami, "High-radix multipliers," in *Computer Arithmetic: Algorithms and Hardware Design*, 2nd ed. New York, NY, USA: Oxford Univ. Press, 2010, pp. 157–171, ch. 10.

[33] O. Hasan and S. Kort, "Automated formal synthesis of Wallace tree multipliers," in *Proc. 50th Midwest Symp. Circuits Syst.*, 2007, pp. 293–296.

[34] J. Fadavi-Ardekani, "$M \times N$ booth encoded multiplier generator using optimized Wallace trees," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 1, no. 2, pp. 120–125, Jun. 1993.

[35] S.-R. Kuang, J.-P. Wang, and C.-Y. Guo, "Modified booth Multipliers with a regular partial product array," *IEEE Trans. Circuit Syst.*, vol. 56, no. 5, pp. 404–408, May 2009.

[36] F. Elguibaly, "A fast parallel multiplier-accumulator using the modified Booth algorithm," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 47, no. 9, pp. 902–908, Sep. 2000.

[37] W.-C. Yeh and C.-W. Jen, "High-speed booth encoded parallel multiplier design," *IEEE Trans. Comput.*, vol. 49, no. 7, pp. 692–701, Jul. 2000.

[38] J. M. Rabaey, A. P. Chandrakasan, and B. Nikolic, "Designing arithmatic building blocks," in *Digital Integrated Circuits: A Design Perspective* (Prentice-Hall Electronics and VLSI Series), 2nd ed. Upper Saddle River, NJ, USA: Pearson Educ., 2003, pp. 559–662, ch. 11.

[39] M. Ha and S. Lee, "Multipliers with approximate 4–2 compressors and error recovery modules," *IEEE Embedded Syst. Lett.*, vol. 10, no. 1, pp. 6–9, Mar. 2018, doi: 10.1109/LES.2017.2746084.

[40] C.-H. Chang, J. Gu, and M. Zhang, "Ultra low-voltage low-power CMOS 4–2 and 5–2 compressors for fast arithmetic circuits," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 51, no. 10, pp. 1985–1997, Oct. 2004.

[41] PASCAL. *The Pascal Visual Object Classes Homepage*. Accessed: Apr. 2019. [Online]. Available: http://host.robots.ox.ac.uk/pascal/VOC/index.html

[42] R. A. Solovyev, A. A. Kalinin, A. G. Kustov, D. V. Telpukhov, and V. S. Ruhlov, "FPGA implementation of convolutional neural networks with fixed-point calculations," 2018. [Online]. Available: arxiv.abs/1808.09945.

[43] T. B. Preußer, G. Gambardella, N. J. Fraser, and M. Blott, "Inference of quantized neural networks on heterogeneous all-programmable devices," in *Proc. Design Autom. Test Eur. Conf. Exhibit.*, Mar. 2018, pp. 833–838.

[44] Y. J. Wai, Z. B. M. Yussof, S. I. B. Salim, and L. K. Chuan, "Fixed point implementation of Tiny-YOLO-V2 using OpenCL on FPGA," *Int. J. Adv. Comput. Sci. Appl.*, vol. 9, no. 10, pp. 506–512, 2018.

[45] H. Nakahara, H. Yonekawa, T. Fujii, and S. Sato, "A lightweight YOLOv2: A binarized CNN with a parallel support vector regression for an FPGA," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, Feb. 2018, pp. 31–40.

[46] D. T. Nguyen, T. N. Nguyen, H. Kim, and H. Lee, "A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 8, pp. 1861–1873, Aug. 2019, doi: 10.1109/TVLSI.2019.2905242.

[47] B. Liu, D. Zou, L. Feng, S. Feng, P. Fu, and J. Li, "An FPGA-based CNN accelerator integrating depthwise separable convolution," *Electronics*, vol. 8, no. 3, p. 281, 2019.