JOURNAL OF BASIC SCIENCE AND ENGINEERING

Design of a optimized and High Performance MAC for CNNs

[1] Manjula B Bhajantri, [2] Dr.Sharanabasaveshwar G Hiremath

[1] Research Scholar, Department of Electronics and Communication Engineering, East West Institute of Technology, Bangalore, Visvesvaraya Technological University, Belagavi, Karnataka.
[2] Professor, Electronics and Communication Engineering, East West Institute of Technology, Bangalore, Visvesvaraya Technological University, Belagavi, Karnataka.

DOI: https://doie.org/10.0618/Jbse.2024912129

Abstract

The latest Convolutional Neural Networks (CNNs) incorporate a greater number of convolution layers compared to previous iterations, aiming to enhance classification accuracy and super-resolution. While numerous researchers concentrate on minimizing network size to lower computational costs while maintaining accuracy, others explore optimizing individual convolution layers to reduce computational expenses. In this paper, we introduce a novel approach to approximate computing utilizing 4-2 compressors, applied to both Baugh Wooley and Booth multipliers. Convolution layers within CNNs typically employ multiply-and-accumulate (MAC) operations. We have integrated approximate multipliers into the adapted MAC structure to enhance the efficient utilization of Field Programmable Gate Array (FPGA) resources. Our results demonstrate that the proposed approximate compressors exhibit a reduction of 15.4% in the area-delay product (ADP) and 35.7% in the area-power product (APP) compared to previous design methodologies.

Keyword: CNN, FPGA, MAC, Multiplier, Approximate Compressor

1. Introduction

Y. LeCun et al. introduced the Convolutional Neural Network (CNN) [1, 2], which has gained increasing popularity due to its superiority in tasks such as image recognition, classification, and speech recognition compared to other deep learning algorithms. CNN can be delineated into two main components. Firstly, there is the training phase where, for instance, it learns the distinctive features of each image within the database to facilitate image classification.

The subsequent phase involves classification. Here, CNN scrutinizes the features of a new input image and endeavors to correlate them with the training data, thereby determining the nature of the image. We will delve deeper into the intricacies of CNN in the subsequent section. Accurate image classification holds paramount importance. To enhance recognition accuracy, CNN necessitates additional convolution layers. For instance, AlexNet, depicted in Figure 1, which encompasses five convolution layers, clinched the top prize at the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) with a top-5 error rate of 15.3% [3]. Nonetheless, ResNet, the victor of ILSVRC 2015, augmented the layer count to 152 and attained a Top-5 error rate of 5.71% [4]. Table I delineates the Top-5 error rates from 2012 to 2015 at the ILSVRC, showcasing a substantial reduction as the layer count escalates. This underscores the indispensability of augmenting the convolution layers for heightened accuracy in CNN networks. However, when CNNs are integrated into hardware systems such as CPUs, GPUs, and Field Programmable Gate Arrays (FPGAs), the hardware expenses surge in direct correlation with the layer count. The reason lies in the intensive computational demands of convolutional layers, while fully connected (FC) layers require substantial memory resources [5]. To alleviate the computational burden of CNNs, diverse methodologies have been explored. Y. Gal et al. [30] proposed a Bayesian CNN approach tailored for small datasets, achieved by imposing a probability distribution over the CNN's kernel. X. Lin et al. [15] investigated binary CNNs, where weight and bias values are confined to {-1, +1} during runtime, resulting in a significant reduction in memory usage and computational expenses.

Hardware accelerators encompass a variety of types of hardware, with Application-specific Integrated Circuits (ASICs) and FPGAs emerging as the most prevalent choices. Employing ASICs as hardware accelerators typically offers greater efficiency compared to FPGAs. Nevertheless, FPGAs boast greater flexibility, accommodating a wider range of applications than ASICs. Consequently, FPGA environments are gaining popularity owing to their superior energy-to-performance efficiency [7, 8]. Furthermore, recent research indicates that the performance of FPGA systems is comparable to that of CPUs and GPUs [9]. However, the primary weakness of FPGA systems lies in their extremely limited resources. To perform Multiply-Accumulate (MAC) operations, FPGA utilizes Digital Signal Processing (DSP) and Look-Up Table (LUT) blocks. Typically, only DSPs are employed for multiplication operations of dimensions × and b. As these operations are executed in parallel, the number of MAC operations corresponds to the number of DSPs utilized in the FPGA. D. Nguyen et al. [29] introduced the concept of double MAC, which effectively reduces the number of DSPs required by half. Nonetheless, this approach proves beneficial only for scenarios involving small bit sizes, such as 4 × 4 multiplications. Once the calculations surpass the capacity of a single DSP (e.g., exceeding 27 × 18 on the DSP48E2 within the Xilinx ZCU102 FPGA board), additional resources become necessary. For instance, two DSPs or one DSP along with supplementary LUTs are needed for computation. Z. Zhang et al. [30] suggested an alternative by amplifying the number of MAC operations simultaneously. In cases where DSP resources prove insufficient to accommodate all required multiplications within the network, LUTs are employed for the remaining multiplications, albeit with less efficiency.

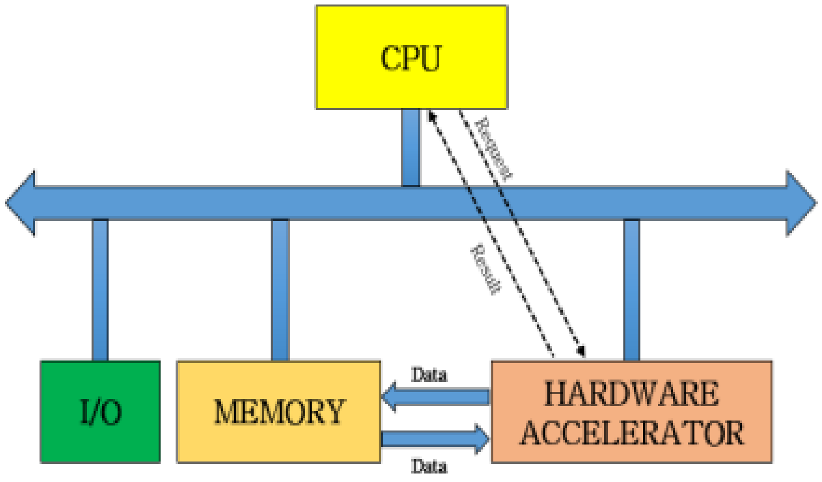


Figure 1: Off-Chip Hardware Accelerator Architecture

A multiplier comprises three key components: partial product generation, partial products reduction, and final addition through carry-propagate adders. Approximation techniques can be integrated into any of these stages. For instance, in [16], approximation methods are applied during the partial product generation phase. Another approach, known as partial product perforation, which involves omitting the generation of certain partial products, is explored in [17]. The efficacy of these techniques is assessed across different multiplier implementations, with power consumption and error rates compared for analysis. Meanwhile, during the partial products reduction phase, various types of adders, including half/full adders and 4-2 compressors, are employed. Much of the prior research on approximate multipliers has concentrated on this particular stage. For instance, Esposito et al. [18] introduced an approximate half adder by streamlining a 2-1 compressor (half-adder) with an OR gate. By achieving a mere 1/16 error probability, the full adder can be simplified using just one AND gate and two OR gates. Additionally, 4-2 compressors are frequently utilized to further reduce the partial product step compared to half/full adders. Another method for approximate multipliers is outlined in [19], where large multiplications are partitioned into smaller blocks, such as 2×2 multiplier blocks, and adjustable multipliers are constructed. Truncation methods have also garnered significant attention. For instance, C. Chang et al. [20] employed a multiplexer-based truncated array multiplier to diminish power consumption and area utilization. Achieving a balance between accuracy and computational cost, encompassing power consumption and area utilization, is crucial in implementing multipliers. Thus, it becomes imperative to ascertain the optimal equilibrium between these factors. This paper introduces low-power and area-efficient multipliers by proposing and employing novel approximate 4-2 compressors. Furthermore, the conventional Multiply-Accumulate (MAC) operation is adapted to perform two operations concurrently. The application of Convolutional Neural Networks (CNNs) in VDSR is utilized to validate the efficacy of these MAC operations.

1.1 CNN Architecture

A CNN architecture comprises input and output layers, with hidden layers sandwiched in between, as illustrated in Figure 2. These hidden layers play a pivotal role in CNNs as they directly impact the accuracy of the model's outputs. Typically, the hidden layers consist of a sequence of components including convolutional layers (CONV layers), activation functions such as Rectified Linear Unit (ReLU), pooling layers, and fully connected (FC) layers.

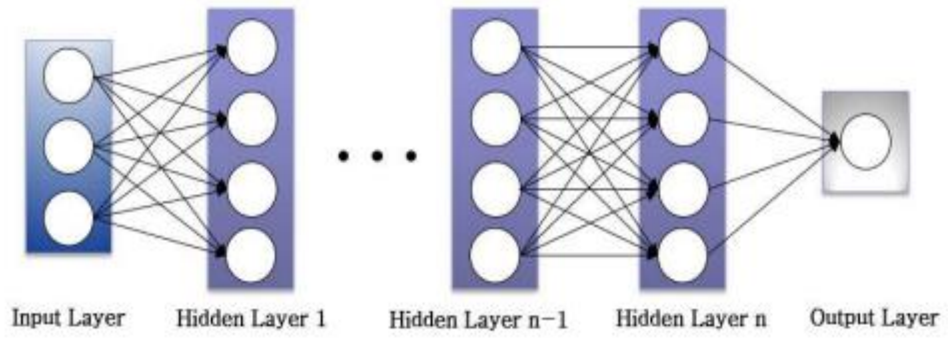


Figure 2: A Regular n-layer Neural Network

CONV layers perform computations involving inputs, weights, and biases. These weights are commonly referred to as kernels. Illustrated in Figure 3, when inputs traverse the kernel, they undergo multiplication with the weights and subsequent accumulation with biases. This process involves matrix multiplications, enabling parallel computation. The number of outputs generated equals the number of kernels utilized in the operation.

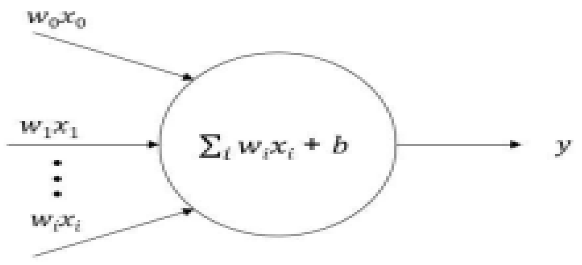


Figure 3: Convolution Calculation

Activation functions are applied to the outputs of kernels. Rectified Linear Unit (ReLU) stands out as a commonly employed activation function due to its ability to mitigate the likelihood of gradient vanishing and its promotion of sparsity. ReLU functions by setting negative values to zero while preserving positive values, as elucidated in (1).

$$h = \max(0, y), \text{ where } y = wx + b \qquad (1)$$

Vol. 21, No. 1, (2024)
ISSN: 1005-093

Pooling is another pivotal concept in CNNs, characterized by its non-linear nature as it serves to decrease the number of features. Among various pooling methods, max pooling is the most prevalent. This operation entails extracting the maximum value from the values within a filter. Figure 4 provides a visual depiction of how max pooling is conducted.

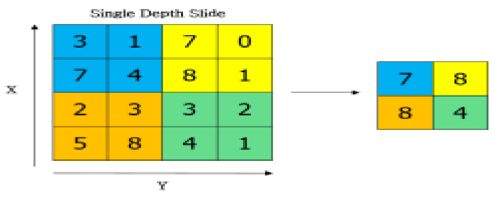


Figure 4: Max Pooling with a 2 × 2 filter and stride = 2

FC layers serve as the output layers in a CNN architecture. Following multiple CONV layers and max pooling layers, FC layers establish connections between every neuron in one layer to every neuron in another layer. The flattened matrix subsequently undergoes processing through an FC layer for classification purposes.

1.2 CNN Hardware Architecture

Figure 5 provides an overview of the CNN hardware architecture. Due to the extensive volume of data, weights are stored in external memory. The initial CONV layer receives input feature map data and weights from this external memory, then performs MAC operations. The outcomes are stored in a buffer, thereby minimizing data communication between the CNN hardware and external memory. Consequently, this setup also contributes to a reduction in power consumption.

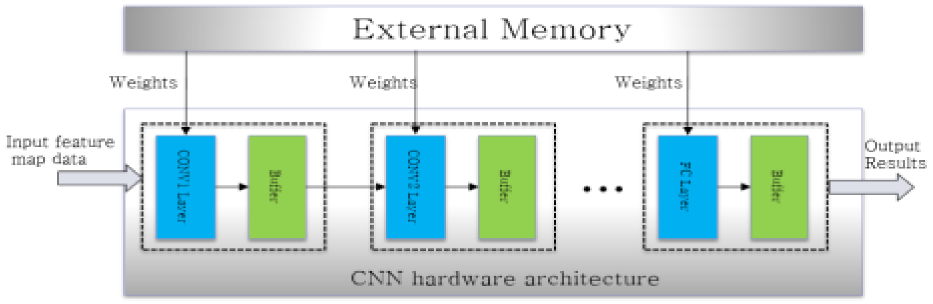


Figure 5: An Overview of CNN Hardware Architecture

CONV layers are recognized as computation-intensive layers, whereas FC layers are identified as memory-intensive layers [14]. This distinction underscores our focus on reducing computational costs primarily within CONV layers. Conversely, FC layers necessitate substantial data retrieval from external memory, highlighting the significance of minimizing the number of weights associated with them.

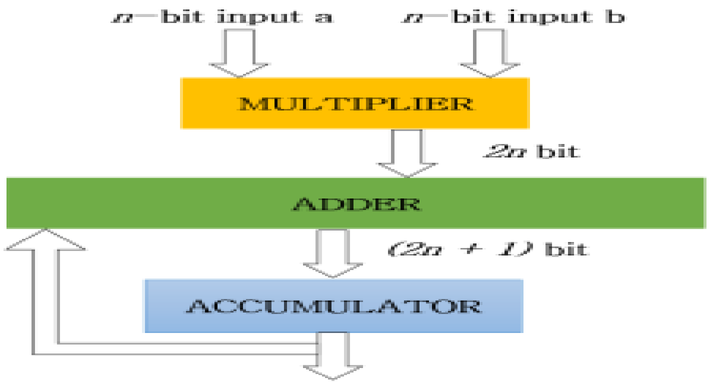


Figure 6: A Conventional MAC Structure

Figure 7 depicts a conventional MAC structure, serving as the computational engine for the CONV layer. Two n-bit inputs are multiplied, resulting in a product of 2n bits. Here, input "a" can be interpreted as input feature map data from the image, while input "b" represents a weight value from the kernel. The multiplied results are subsequently accumulated with other results from the same kernel, utilizing an adder. Upon completion of each operation within the MAC, the results traverse through ReLU before proceeding to the pooling layer for downsizing, as elaborated in the preceding section.

2. Proposed methodology of MAC for CNN

This study aims to offer a comprehensive review and comparative analysis of approximate 4-2 compressors previously proposed in literature. Additionally, we introduce a novel approximate compressor, bringing the total number of analyzed compressors to twelve. These circuits are then utilized in the design of 8 × 8 and 16 × 16 multipliers, implemented in CMOS technology. For each operand size, we examine two multiplier configurations, featuring varying levels of approximations, both signed and unsigned. Our investigation underscores the absence of a singular optimal approximate compressor topology, as the most suitable solution hinges on factors such as the required precision, the signedness of the multiplier, and the error metric under consideration.

The proposed approximate 4-2 compressor leverages the stacking circuit concept, initially introduced in [33] for designing hardware-efficient 6-3 and 7-3 exact compressors, and adapted herein for approximate compressor implementation. With four inputs x1, x2, x3, x4, a 4-bit stacker circuit produces four outputs y1, y2, y3, y4, where y1 is set high if any of the inputs is '1', y2 is set high if any two inputs are '1', and so forth. The behavior of a four-bit stacker is delineated by the following Boolean equations:

$y1 = x1 + x2 + x3 + x4$ ....(1)

y2 = x1x2 + x1x3 + x1x4 + x2x3 + x2x4 + x3x4 ....(2)
y3 = x1x2x3 + x1x2x4 + x1x3x4 + x2x3x4.. (3)
y4 = x1x2x3x4 ....(4)

A. Partial Products Reduction

The Partial Product Matrix (PPM) is minimized through a methodology akin to the Dadda multiplier, which employs multiple stages of compressors to diminish the maximum height of the PPM to two rows. Determining the maximum height involves working backward from the final stage. Utilizing 4-2 compressors yields heights of 2, 4, and 8.

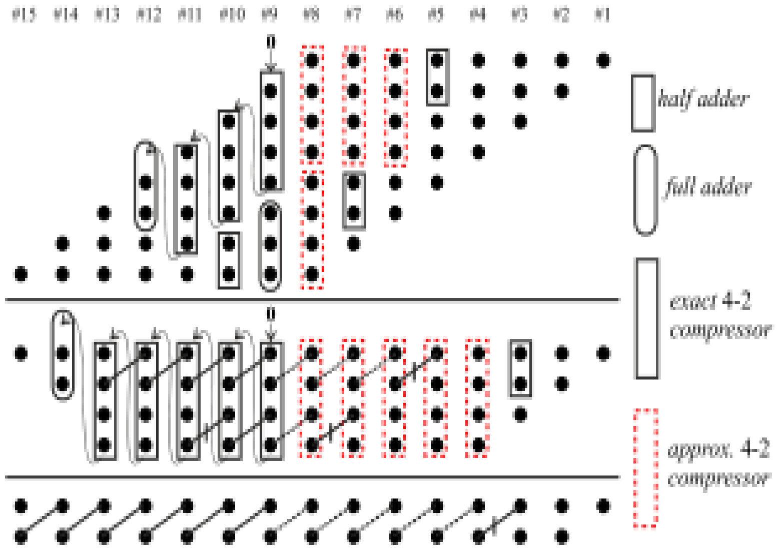


Figure 7: Reduction scheme for an 8 × 8 unsigned multiplier with C − N configuration

Approximate 4-2 compressors exclusively operate within the 8 least significant columns of the partial-product matrix. The Carry and Sum outputs of approximate 4-2 compressors are indicated by dotted lines, while solid lines denote connections for full adders and outputs of exact 4-2 compressors. Half adder outputs are represented by solid lines with a vertical bar.

The reduction process employs heights of 16, 32, and so forth as the maximum thresholds for the various stages. Full and half adders are exclusively utilized when necessary to achieve the requisite reduction of the PPM. As illustrated in Figure 7, which serves as an example for an unsigned 8 × 8 multiplier in a C-N configuration, the diagram showcases the reduction scheme. In this depiction, the Carry and Sum outputs of the approximate 4-2 compressors are depicted with dotted lines, while solid lines represent the outputs of exact 4-2 compressors and full adders. Half adder outputs are linked by solid lines with vertical bars. The arrowed lines depicted in Figure 4 illustrate the connections between the Tin-Tout pins of exact 4-2 compressors. Noteworthy are the full-adders situated in column #12 of the first stage and column #14 of the second stage, essential for receiving the Tout signal from the exact 4-2 compressor in the preceding column. In the C-N configuration, the 8 × 8 multiplier utilizes 9 approximate 4-2 compressors and 8 exact compressors, whereas in the case of the 16 × 16 multiplier, the quantities of approximate and exact compressors are 49 and 48, respectively. It is crucial to note that the examined approximate compressors (excluding Yang1 and Lin) lack input symmetry, as the error is contingent upon the specific order in which the input bits are connected to the x1, x2, x3, and x4 inputsPut differently, the count value can vary when the inputs are permuted, even with the same number of inputs set to '1'. Conversely, the inputs of exact compressors can be rearranged without impacting system functionality. For instance, consider the Ha compressor: for the input x4x3x2x1 = 1100, the count value is one, whereas for the input x4x3x2x1 = 0011, the count value becomes two. Consequently, the error performance of a partial products reduction tree hinges on the specific connection of each signal to every input of the approximate compressors. In order to mitigate the error rate, it is essential to take into account the probabilities associated with the signals driving the inputs of the approximate compressors during the design phase of the partial product compression tree. This aspect has been overlooked in prior literature, rendering the design of the partial product reduction tree a challenging task. Additionally, without precise knowledge of the exact order of connections utilized in previous studies, it becomes challenging to reproduce (and subsequently compare) the results presented in prior research.

3. Results and discussions

A Modified MAC

The conventional Multiply-Accumulate (MAC) structure depicted in Figure 8 processes one input data with one weight. To address the challenge of reducing the DSP usage in FPGA, a double MAC approach was proposed by [29]. However, the applicability of the double MAC method is limited to calculations involving small bit sizes. When the number of bits in the multiplication operation exceeds the DSP48E2 specification, requiring 27 × 18, two DSPs or one DSP with additional LUTs become necessary. Therefore, this paper proposes a modified version of the double MAC method aimed at optimizing FPGA resource utilization.

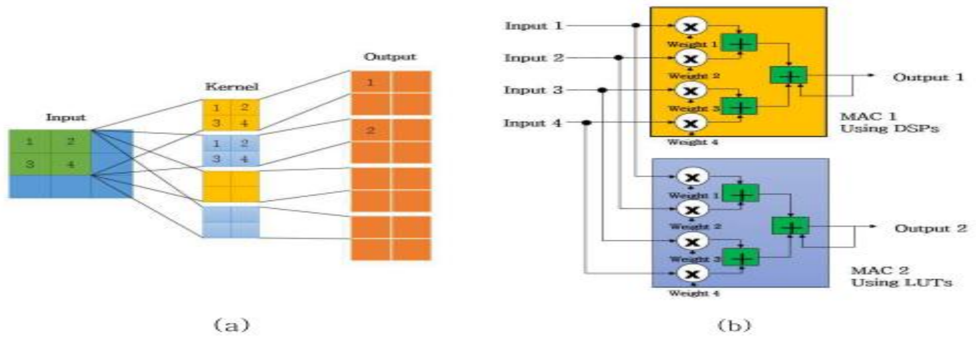


Figure 8: (a) Four 2 × 2 Kernel Convolution (b) A Modified MAC Structure.

Figure 8 presents a modified MAC structure. Figure 15(a) illustrates an example of a four 2 × 2 kernel convolution with a stride of 1. In the conventional MAC setup, DSPs are exclusively utilized for computing one kernel operation. Consequently, a total of 4 DSPs are necessary to process the four 2 x 2 kernel CONV layer. However, for deeper convolutions employed in high-accuracy classification or recognition tasks, the available DSP resources in FPGA may prove inadequate to handle all MAC operations within the CONV layers. In such cases, the FPGA tool automatically converts the remaining MAC operations to LUTs during synthesis, albeit this automatic conversion from DSPs to LUTs by the tool is not an efficient approach. Nonetheless, multipliers, such as Booth and Baugh Wooley multipliers, are well-known for their optimization in terms of area, power consumption, and delay. Furthermore, these multipliers can undergo further optimization through the utilization of approximate compressors. This paper explores the application of approximate compressor designs on Booth and Baugh Wooley multipliers, integrated within a modified MAC setup, aiming for high performance and efficient resource utilization in FPGA environments.

3.1 VDSR Hardware Structure

CNN finds applications in various domains. This paper implements the modified MAC on VDSR, as depicted in Figure 16. A 256 pixel × 256 pixel input image from the host PC undergoes processing through 12 CONV layers on the FPGA board. With input data comprising 14 bits, the previous multipliers are extended accordingly. Furthermore, in line with common CNN practices, the outputs from MAC operations are truncated to 14 bits. Both input image and weight data are stored in the DRAM within the FPGA. The ARM CPU embedded within the FPGA board receives commands from the host PC, facilitating tasks such as data transfer and CNN execution on the FPGA. Upon receiving the 'Run' command from the host, the FPGA initiates the CNN execution for super-resolution tasks. Intermediate data from the CONV layers is stored in on-chip SRAM, functioning as a buffer for the CNN process.

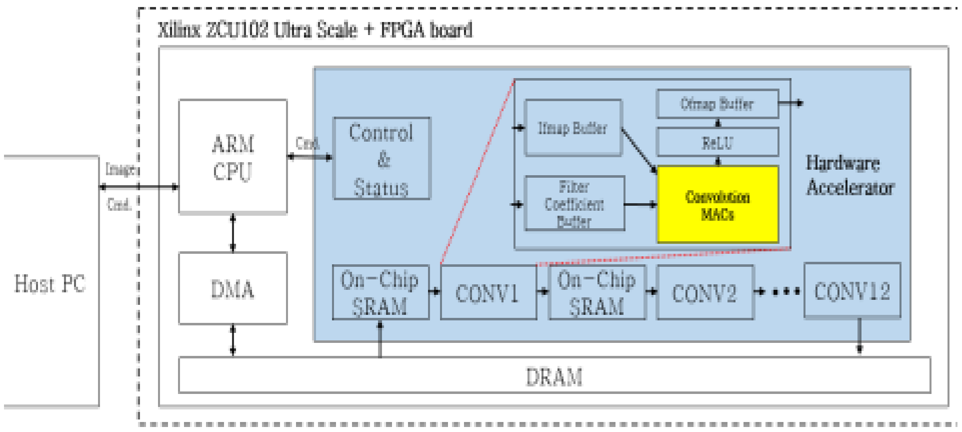


Figure 9: VDSR Hardware Structure

3.2 Multiplier Comparison in CNN application

The adapted MAC finds application within a CNN task, particularly in VDSR, as described in this paper. Figure 9 delineates the experimental arrangement for VDSR. The host PC, running Windows Visual Studio, undertakes quantization of both input image and weight data before transmitting them to the FPGA. Communication between the host PC and FPGA is facilitated by the Xilinx Vivado SDK ver. 2017.03 tool. Subsequently, the host PC transmits the quantized image and weight data to the FPGA, where they are stored in DRAM for further processing. Upon the availability of all required data, the host triggers the initiation of CNN execution on the FPGA. VDSR employs 14 x 14 multiplications for its CNN operations. The input 14 bits comprise 1 signed bit, 3 integer bits, and 10 fraction bits. Initially, approximate compressors are applied to half of the partial products, which are represented as 14 bits. Subsequent experiments involve variations in multipliers, compressors, and the number of applied bits. Specifically, an approximate compressor is applied to 10 out of 28 bits, representing half of the fraction bits.

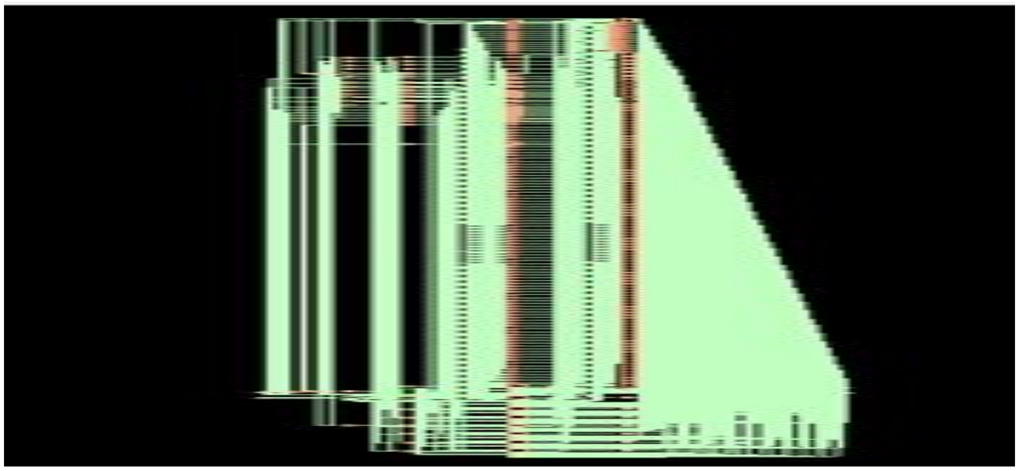
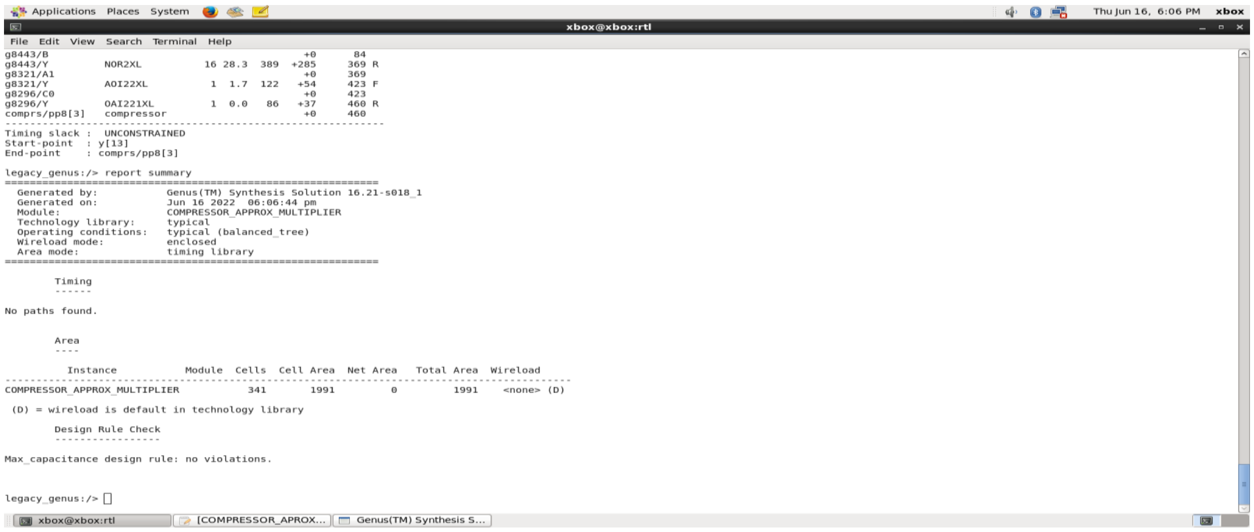
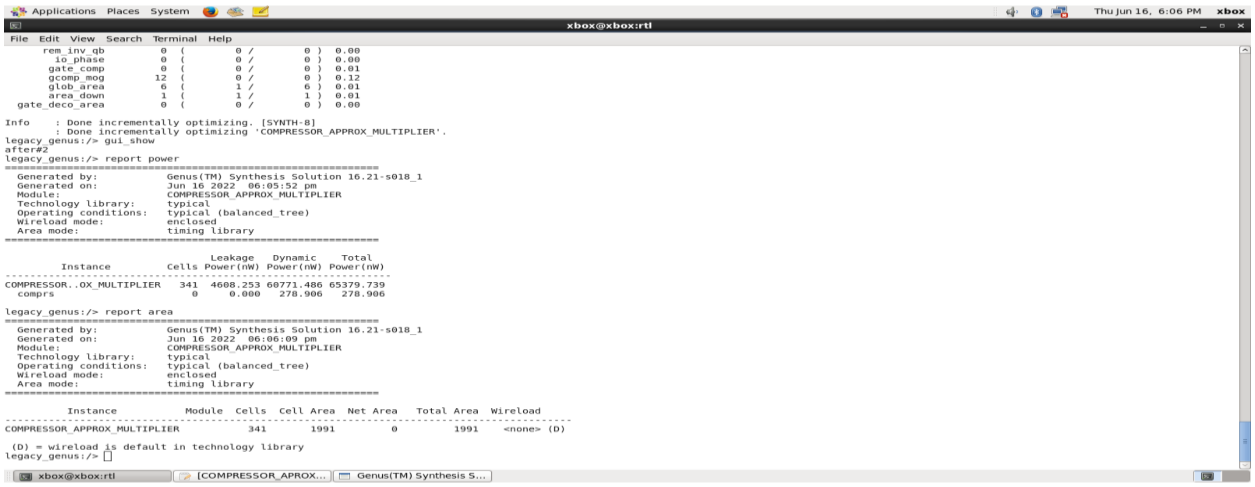


Figure 10: Synthesis results



```
g8443/B                                      +0       84
g8443/Y         NOR2XL          16 28.3  389  +285      369 R
g8321/A1                                     +0      369
g8321/Y         AOI22XL          1  1.7  122   +54      423 F
g8296/C0                                     +0      423
g8296/Y         OAI221XL         1  0.0   86   +37      460 R
comprs/pp8[3]   compressor                   +0      460
------------------------------------------------------------
Timing slack :  UNCONSTRAINED
Start-point  : y[13]
End-point    : comprs/pp8[3]

legacy_genus:/> report summary
============================================================
  Generated by:           Genus(TM) Synthesis Solution 16.21-s018_1
  Generated on:           Jun 16 2022  06:06:44 pm
  Module:                 COMPRESSOR_APPROX_MULTIPLIER
  Technology library:     typical
  Operating conditions:   typical (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

        Timing
        ------

No paths found.

        Area
        ----

          Instance           Module  Cells  Cell Area  Net Area   Total Area  Wireload
------------------------------------------------------------------------------------
COMPRESSOR_APPROX_MULTIPLIER           341       1991         0         1991    <none> (D)

 (D) = wireload is default in technology library

        Design Rule Check
        -----------------

Max_capacitance design rule: no violations.


legacy_genus:/> ▯
```

Figure 11: area report



```
      rem_inv_qb         0  (         0 /         0 )  0.00
        io_phase         0  (         0 /         0 )  0.00
       gate_comp         0  (         0 /         0 )  0.01
       gcomp_mog        12  (         0 /         0 )  0.12
       glob_area         6  (         1 /         6 )  0.01
       area_down         1  (         1 /         1 )  0.01
  gate_deco_area         0  (         0 /         0 )  0.00

Info    : Done incrementally optimizing. [SYNTH-8]
        : Done incrementally optimizing 'COMPRESSOR_APPROX_MULTIPLIER'.
legacy_genus:/> gui_show
after#2
legacy_genus:/> report power
============================================================
  Generated by:           Genus(TM) Synthesis Solution 16.21-s018_1
  Generated on:           Jun 16 2022  06:05:52 pm
  Module:                 COMPRESSOR_APPROX_MULTIPLIER
  Technology library:     typical
  Operating conditions:   typical (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

                                  Leakage   Dynamic    Total
         Instance          Cells Power(nW) Power(nW) Power(nW)
---------------------------------------------------------------
COMPRESSOR..OX_MULTIPLIER   341  4608.253 60771.486 65379.739
  comprs                      0     0.000   278.906   278.906

legacy_genus:/> report area
============================================================
  Generated by:           Genus(TM) Synthesis Solution 16.21-s018_1
  Generated on:           Jun 16 2022  06:06:09 pm
  Module:                 COMPRESSOR_APPROX_MULTIPLIER
  Technology library:     typical
  Operating conditions:   typical (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

          Instance           Module  Cells  Cell Area  Net Area   Total Area  Wireload
------------------------------------------------------------------------------------
COMPRESSOR_APPROX_MULTIPLIER           341       1991         0         1991    <none> (D)

 (D) = wireload is default in technology library
legacy_genus:/> ▯
```

Figure 12: power report

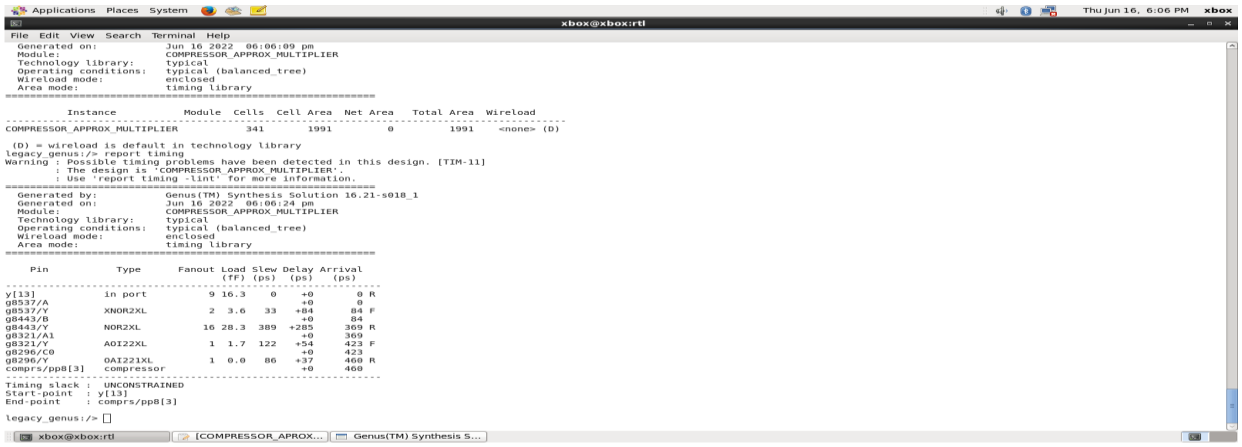

```
  Generated on:            Jun 16 2022  06:06:09 pm
  Module:                  COMPRESSOR_APPROX_MULTIPLIER
  Technology library:      typical
  Operating conditions:    typical (balanced_tree)
  Wireload mode:           enclosed
  Area mode:               timing library
============================================================

          Instance           Module  Cells  Cell Area  Net Area   Total Area  Wireload
-----------------------------------------------------------------------------------------
COMPRESSOR_APPROX_MULTIPLIER            341       1991         0         1991    <none> (D)

 (D) = wireload is default in technology library
legacy_genus:/> report timing
Warning : Possible timing problems have been detected in this design. [TIM-11]
        : The design is 'COMPRESSOR_APPROX_MULTIPLIER'.
        : Use 'report timing -lint' for more information.
============================================================
  Generated by:            Genus(TM) Synthesis Solution 16.21-s018_1
  Generated on:            Jun 16 2022  06:06:24 pm
  Module:                  COMPRESSOR_APPROX_MULTIPLIER
  Technology library:      typical
  Operating conditions:    typical (balanced_tree)
  Wireload mode:           enclosed
  Area mode:               timing library
============================================================

    Pin           Type      Fanout Load Slew Delay Arrival
                                   (fF) (ps)  (ps)   (ps)
------------------------------------------------------------
y[13]           in port          9 16.3    0    +0       0 R
g8537/A                                         +0       0
g8537/Y         XNOR2XL          2  3.6   33   +84      84 F
g8443/B                                         +0      84
g8443/Y         NOR2XL          16 28.3  389  +285     369 R
g8321/A1                                        +0     369
g8321/Y         AOI22XL          1  1.7  122   +54     423 F
g8296/C0                                        +0     423
g8296/Y         OAI221XL         1  0.0   86   +37     460 R
comprs/pp8[3]   compressor                      +0     460
------------------------------------------------------------
Timing slack :  UNCONSTRAINED
Start-point  : y[13]
End-point    : comprs/pp8[3]

legacy_genus:/> 
```

Figure 13: Delay report

Table I: Synthesized in FPGA & Cadence Results

Type of synthesis	Power	Area	Delay
FPGA-4-2 compressors	**4.23 (Watts)**	**24.4 (µm$^{2)}$**	**16.81(ns)**
CADENCE-4-2 compressors	**65379.739 (nw)**	**1991 (nm)**	**460(ps)**

Conclusion

The application of two novel 4-2 compressors on a Booth multiplier demonstrates superior performance in optimized area and power for suitable applications. The comprehensive summary of experimental outcomes results is detailed in the accompanying table. Interestingly, BW multipliers better Booth multipliers in terms of area, delay, and power consumption. However, despite their advantages, super-resolution image results using BW multipliers fail to meet qualification standards. This shortfall can be attributed to the increased number of partial product groups in BW multipliers, necessitating additional compressors to streamline circuitry and achieve satisfactory outcomes.

References

1) Y. LeCun et al., "Backpropagation Applied to Handwritten Zip Code Recognition," in Neural Computation, vol. 1, no. 4, pp. 541-551, Dec. 1989.
2) Yann LeCun and Yoshua Bengio. 1998. Convolutional networks for images, speech, and time series. In The handbook of brain theory and neural networks, Michael A. Arbib (Ed.). MIT Press, Cambridge, MA, USA 255-258.

3) A. Krizhevsky, I. Sutskever, G. E. Hinton, “ Imagenet classification with deep convolutional neural networks, ” Advances in Neural Information Processing Systems, vol. 2, pp. 1097-1105, 2012.
4) K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In CVPR, 2016.
5) J. Qiu et al., “Going deeper with embedded fpga platform for convolutional neural network,” in FPGA. ACM, 2016, pp. 26– 35.
6) Kestur, Srinidhi & Davis, John & Williams, Oliver. (2010). BLAS comparison on FPGA, CPU and GPU. Proceedings - IEEE Annual Symposium on VLSI, ISVLSI 2010. 1. 288-293.
7) E. Nurvitadhi, J. Sim, D. Sheffield, et. al., “ Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC,” Field Programmable Logic and Applications (FPL), 2016.
8) S. Che, J. Li, J. W. Sheaffer, K. Skadron and J. Lach, "Accelerating Compute-Intensive Applications with GPUs and FPGAs," 2008 Symposium on Application Specific Processors, Anaheim, CA, 2008, pp. 101-107.
9) E. Nurvitadhi, G. Venkatesh, J. Sim, et. al., “Can FPGAs Beat GPUs in Accelerating Next-Generation Deep Neural Networks?” International Symposium on Field-Programmable Gate Arrays (ISFPGA), 2017.