Implementation of Convolutional Neural Network with Co-design of High-Level Synthesis and Verilog HDL

Hejie Yu, Jun Cheng, Xiangnan Zhang, Yuzhe Gao, Kuizhi Mei

School of Microelectronics, Xi'an Jiaotong University, Xi'an 710049, China
* Email: meikuizhi@mail.xjtu.edu.cn

Abstract
In recent years, Convolutional Neural Networks(CNNs) have been widely adopted for image classification and target recognition. As one of CNN's main hardware implementation platforms, FPGA has its advantages of high flexibility, excellent trade-off between performance and power, but still has the problems of complex developing processes and poor adaptability for various algorithm models. Therefore, a high-performance and fast hardware implementation architecture adaptation to the CNNs is presented in the paper. The hardware architecture is designed with co-design of High-Level Synthesis(HLS) and Verilog HDL, which simplifies the design process and ensures performance. And it adopts the variables parameterization to deal with the problem of pool model adaptability. With row-by-row calculation, the circuits adopt the layered parallelism to ensure the flexibility of convolution and the pipeline parallel calculation to improve the speed. The paper achieves an overall average 359.7GOP/s for the AlexNet on Xilinx ZCU104 platform.

1. Introduction

Now, CNN has become the most mainstream algorithm with the advantages of high recognition accuracy and convenient feature extraction and widely used in the fields of target detection, object classification, and automatic driving. The actual application of the CNN algorithm needs to be implemented on an embedded hardware platform. The current hardware platforms for deep learning algorithms mainly consist of GPU, ASIC and FPGA. Compared with GPU, FPGA has the advantages of low latency, high performance and low power consumption, which is more suitable for the inference implementation of CNN algorithm models. Compared with ASIC, FPGA has high flexibility, low cost, and short design periods, which can be used for specific field. Thus implementation of CNNs on FPGAs has become the current research focus, but it also faces a series of problems, including: how to use limited FPGA resources to meet the computing and data storage needs of multiple kinds of neural networks and get higher performance, how to simplify the FPGA hardware design process. To solve these problems, the main research goal of this paper is to design a low-power, high-performance and fast hardware implementation architecture suitable for convolutional neural networks, and deployment of the CNNs is implemented to the FPGA platform under the deep learning framework.

2. Hardware architecture design

2.1 Hardware architecture design methods

Both the traditional design method based on HDL and the HLS design method have their own advantages and disadvantages. Thus the paper adopts the collaborative design method of HLS and Verilog HDL to simplify the design process while ensuring design efficiency, and maximize the benefits of hardware performance and design overhead. The HLS design programming with the C language is adopted to implement computationally intensive algorithm modules, including the convolution module, the LRN module. This facilitates the writing and modification of codes and simplifies the design process and period. The lower-level circuits design programming with Verilog HDL, is mainly used to realize network inference data flow control and calculation resource scheduling, including data transfer buffer design, data flow scheduling, system control. The modules are not very difficult to design but have high performance requirements. As a result, they are suitable for RTL-Level circuit. Not only can they fine-tune the integrated circuit structure, but also facilitate timing optimization and take advantages of high-performance design of the underlying hardware.

For adaptation to the diversity of CNN algorithm models, the solution to hardware circuits is the corresponding reconfigurability. The design of the paper only starts from the perspective of hardware and adapts to one specific fixed network model that has been trained for inference. Therefore, we only pay attention to the differences in the structure of different models, and adopt the key variables parameterization design method to adjust the network model and layer structure. The actual implementation includes: the overall circuit recons -truction configuration and partial circuit reconstruction configuration. When the overall circuit is reconstructed, there are many parameters need to be configured, and we can manually adjust the codes or modify the parameters by encoding and decoding through processor instructions. When the partial circuit is reconstructed, the amount of change is small, and it is only need to use the automatic algorithm of flag variables to switch.

978-1-7281-6235-5/20/$31.00 ©2020 IEEE

2.2 Overall hardware system structure

The overall structure of the system is shown in Figure 1, which includes network modules and external interface circuits. The network modules mainly realize the reconstruction of the model and data calculation, including the basic layer structure circuits (convolution module, activation function, pooling module and LRN module) and the state machine control circuits. The calculation is performed in serialization between layers, and the entire module is time-multiplexed to complete the layer by layer calculation, which can effectively save hardware resources. The external interface circuits mainly realize the data transfer among the network modules, the ARM processor and DDR4 memory.

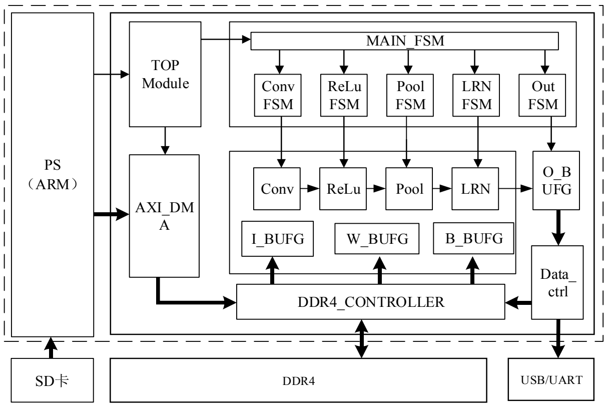


Figure 1. Overall structure of hardware system

2.1.1 Convolution Module

The convolution circuit is a calculation module with row output priority. The neuron is calculated row-by-row and the output result buffer accumulation. The corresponding parallelization calculation design is carried out, which is the combination of four parallelisms:

Neurons Internal Parallelism. In row-by-row calculation, it is changed from full parallel to partial parallel, and the parallelism of the convolution kernel of r×r is changed from r^2 to r. In this way, the calculation time is increased by r times, and the circuit area is reduced by r times.

Parallelism among Neurons. It is realized on the basis of complete parallel Internal Neurons, so the degree of parallelism among neurons is 1.

Parallelism among the input feature maps. The design instantiates the FIFOs to buffer the intermediate accumulation results. In order to avoid repeated accumulation and increasing the resources consumption, parallelism of the input feature map is fully designed to M.

Parallelism among the output feature maps. It is L which requires flexible consideration to ensure that the total parallelism obtained by the product of the four parallelisms satisfies not exceeding the number of instantiable buffer resources and DSP units in the FPGA.

Convolution IP designed with HLS calls the 'for' loop function to complete calculation. As shown in Figure 2, four loops are used to represent four kinds of parallelism calculations, where loop1 and loop2, loop3 and loop4 are combined to form a two-stage pipeline through pipeline optimization instructions. As shown in Table 1, for different layers, layered parameterization parallelism is used to achieve high calculation speed with limited resources.

```
loop2:for(i=0;i<(r×M×L);i++)
  loop1: for(j=0;j<(M×L);j++)
            {out_temp[j]=out_temp[j]+data[i]×weight[i];}

loop4:for(l=0;l<L;l++)
  loop3: for(k=0;k<M;k++)
            {out [l]= out [l]+out_temp[l*M+k];}
```

Figure 2. Convolution calculation pseudocode

Table 1. Layered parallelism for AlexNet

	r	M	L	total parallelism
Conv1	11	3	16	528
Conv2	5	48	4	960
Conv3	3	256	1	768
Conv4	3	192	1	576
Conv5	3	192	1	576

The flow of convolution calculation is shown in Figure 3. Assume that the input image size is W×H×M and the filter size is r×r×M×N. Each calculation, the size of the input image is 1×r×M and the size of the output feature map is 1×1×L, r×M×L PE units need to be instantiated. After completing one calculation, the image window moves according to the step size to Re-read the image data until the end of one row. The size of the output feature image is 1×C×N. The calculation results are saved in the FIFOs, then window moves to read the image data of the second row and repeat the calculation to get the accumulation result until the final output result is obtained after the r-th row data operation is completed. After reading the data in the FIFOs and adding the offset, it will be sent to the activation function module to get the results of one row in the output feature image. The convolution module continues to accept the next row's image data for convolution calculation, until all convolution calculations are completed.

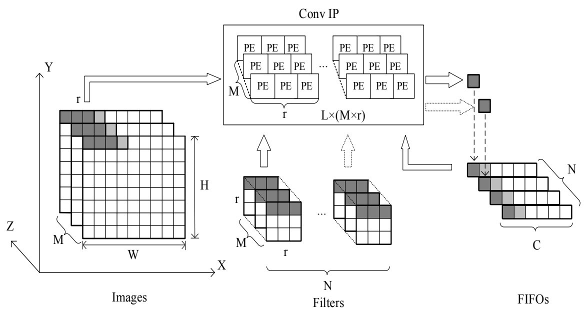


Figure 3. Convolution calculation process

In the specific calculation, two pipeline structures are established. Figure 4 shows the convolution calculation and data input pipeline. Converting this convolution and the next data input from an independent sequence to a simultaneous process can speed up the calculation.

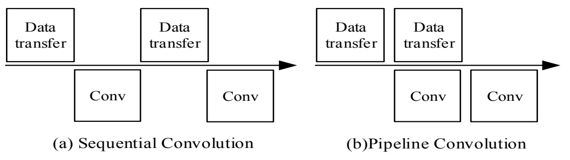


Figure 4. Comparison of convolution calculation sequence structure and pipeline structure

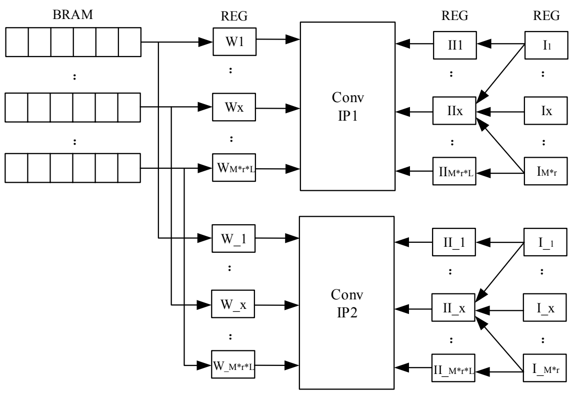
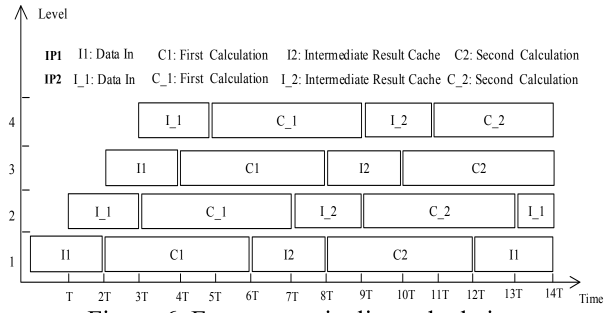


Figure 5. Four-stage pipeline structure consists of dual register banks + dual convolution IPs

Often limited by the BRAM resources in FPGA, the parallelism cannot be further improved, resulting in a low DSP utilization rate. Therefore, the design uses the dual register banks + dual convolution IPs to form a pipeline calculation and construct a four-stage pipeline structure as shown in Figure 5. In this way, the utilization of DSP resources can be increased by 2 times. The specific calculation process is shown in Figure 6. Compared with the previous two-stage pipeline structure, the calculation speed can be doubled.


Level
IP1 I1: Data In C1: First Calculation I2: Intermediate Result Cache C2: Second Calculation
IP2 I_1: Data In C_1: First Calculation I_2: Intermediate Result Cache C_2: Second Calculation
4
3
2
1
I_1
C_1
I_2
C_2
I1
C1
I2
C2
T
2T
3T
4T
5T
6T
7T
8T
9T
10T
11T
12T
13T
14T
Time
Figure 6. Four-stage pipeline calculation

2.1.2 Activation Function and Pooling Module

The design of the RELU activation function circuit is simple, and it can be realized with the pooling circuit in the specific implementation. The pooling circuit realizes a maximum pooling calculation through the recycling comparison with partial buffer results.

2.1.3 LRN Module

The design of the LRN module circuit realizes the data normalization among the feature maps, and the core calculation formula for AlexNet is:

$$X^{\epsilon} = X \Big/ (1 + \alpha \sum_{i}^{n} X_i^2)^{\beta} \quad (\alpha = 0.0001/5,\ \beta = 0.75) \tag{1}$$

Stepwise parallel pipeline calculation is applied in LRN module:

(1) The calculation process is implemented in steps to simplify the calculation and design difficulty;

(2) Instantiating multiple LRN module units (16 or 4 for AlexNet) can improve the calculation speed;

(3) For a long normalization process, the pipeline calcula -tion can accelerate computing.

3. Experimental results

On the ZCU104 platform, performance tests on CNN models as AlexNet are performed. The peak computing power and average computing power of AlexNet respecti -vely reach 515.3GOP/s and 359.7GOP/s at 200MHz frequency. Compared with the previous design in Table 1, the performance of the fast hardware implementation architecture design adaptation to CNNs is quite better.

Table 2. Performance comparison for AlexNet

	[1]	[2]	[3]	Our Impl
Precision	32bit float	16bit fixed	Q15	16bit fixed
Device	VX485T	GSD8	XC7Z045	XCZU7
Frequency(MHz)	100	120	150	200
DSP	2800	1963	900	1728
BRAM(kb)	2060×18	2567×20	1090×18	624×18
Conv1 (GOP/s)	27.5	-	28.6	180.6
Conv2 (GOP/s)	83.8	-	38.4	444.4
Conv3 (GOP/s)	78.8	-	38.4	515.3
Conv4 (GOP/s)	77.9	-	35.4	398.3
Conv5 (GOP/s)	77.6	-	38.4	389.6
Conv aveage(GOP/s)	61.62	72.4	35.84	359.7
Power(W)	18.6	19.1	<10	9.5
Energy efficiency (GOP/s/W)	3.31	3.79	3.58	37.9

4. Summary

Aiming at the problems of complex development process and poor models adaptability in FPGA implementation of CNN, the paper presents a high-performance and fast hardware implementation architecture that adapts to CNNs. At 200MHz frequency, the average computing power of AlexNet reaches 359.7GOP/s on the ZCU104 platform.

References

[1] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, in FPGA, 2015.

[2] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.-s. Seo, and Y. Cao, in FPGA, 2016.

[3] Moini S, Alizadeh B, Emad M, et al. IEEE Transactions on Circuits & Systems II Express Briefs, 2017, 64(10):1-1.