Kernel Sharing Winograd Systolic Array for CNN Acceleration

Group 9: Wenjie Geng, Taoran Ji, Jason Ning, Yuxi Peng
{wenjieg, taoranji, zyning, pengdd}@umich.edu

1 Introduction

Convolutional Neural Networks (CNNs) are widely used for image processing tasks because of their feature extraction abilities. Current state-of-the-art CNNs primarily use kernel sizes of 3×3 and 1×1. CNNs can achieve high output accuracy, but they are computationally intensive due to the convolution operation. In recent years, Winograd's Minimal Filtering Algorithm [1] has been adopted for CNN acceleration because of its fewer multiplications and opportunity for data reuse compared to conventional methods like the FFT.

We have implemented a configurable Kernel Sharing Winograd Systolic Array [2] that can perform both 3×3 and 1×1sized convolution. Furthermore, we implemented our own data controllers and memory to facilitate data flow and data storage. We were able to achieve throughput of 4 Winograd convolution per cycle, with total area of 0.598 mm^2, and 0.282 W of total power. The design is fully verified against our fixed-point MATLAB model.

2 Background Information

2.1 3d Convolution Terminologies

We define $[ID]$ as the depth of input feature map, $[OD]$ as the depth of the output feature map. Each 3D kernel consists of $[ID]$ 2D kernels (one for each input channel), stacked along the depth dimension. These 2D slices convolve individually with their corresponding slices of the input feature map. The results from these individual convolutions are then summed element-wise across the depth dimension to form the output for one channel in the output feature map.

This process is repeated for all $[OD]$ output channels, where each channel has its corresponding kernel. The final output of the 3D convolution is an output feature map with a depth of $[OD]$, reflecting the number of output channels specified. **Figure 1** illustrates this process.

2.2 Winograd Convolution

Winograd 2d convolution computes an $m \times m$ output matrix Y by performing matrix multiplication on an $(m + k - 1) \times (m + k - 1)$ input matrix d with a $k \times k$ kernel matrix g as described in **Equation 1**. The transform matrices G, B, and A are constants and be generated by the Cook-Toom algorithm [3]. The matrix multiplication with B and A can be done with shift and add operations, and matrix V can be pre-computed with the known kernels.

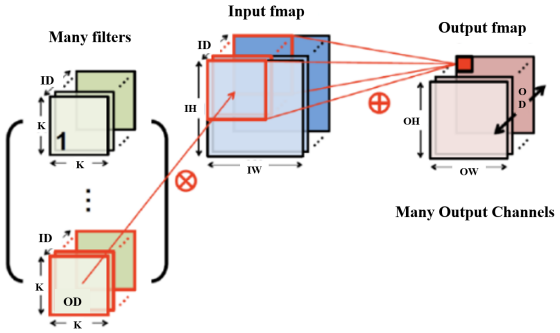

Figure 1: Convolution of 3-dimensional matrix

$$Y = A^T\left[\left(GgG^T\right) \odot \left(B^T dB\right)\right]A \qquad (1)$$
$$Y = A^T\left[V \odot U\right]A$$

A convolutional layer with kernel size $k \times k$ in a CNN can be computed iteratively with the Winograd algorithm with a configuration $F(m \times m, k \times k)$. To achieve support for multiple kernel sizes, we can vary the output size m, so the U, V matrices are size of $w \times w$. From **Figure 2**, we can observe that a large amount of the transform matrices shares the same values if they have the same w. Hence, we can select parts of matrix A and G so we can have a unified architecture for various filter sizes. Note that this method also works for larger kernel sizes such as 5×5 and 7×7.

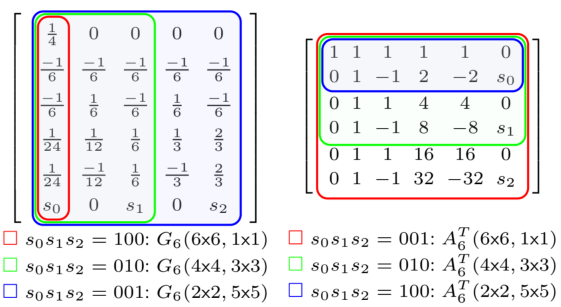

Figure 2: Winograd $w = 6$, transformation matrices for $k = 1, 3, 5$.

2.3 Systolic Array

A systolic array is typically composed of interconnected Processing Elements (PEs). Systolic array architectures are efficient for parallel processing because they enable data reuse, achieving a higher computation throughput without increasing the memory bandwidth. 2d convolution requires constant fetching of input and weight matrices from memory. The challenge for implementing a systolic array architecture for CNNs is the efficient access of external memory.

2.4 Data Pre-processing and Signal Paths

Data pre-processing is a crucial step in machine learning. Techniques like input normalization can enhance model performance by bringing input features into the same scale. In this project, we employed input normalization, zero padding, and fixed-point quantization so an input image can be processed by our VLSI implementation. First, the input image matrix is zero padded according to the kernel size to avoid indexing out-of-bound errors when performing the Winograd convolution. Then the input image and the kernel matrices are normalized, so the values are between -1 and 1. The kernel matrix is multiplied with the G matrix to form the V matrix. The V matrix is quantized to a fixed-point (12,11) representation while the input matrices d are quantized to a fixed-point (8,7) representation. The internal signals inside the PE are in (16,11), and the output m is in (12,7).

3 Proposed VLSI Architecture

3.1 High Level Overview

There are 6 main modules in our Winograd systolic array architecture as shown in **Figure 3**. The memory consists of four partitions, two for storing the input feature map, and the kernel matrices; two for storing the output feature maps. The main controller, data controller, and weight controller work together to fetch data from the SRAM and facilitate inputs to the PE array. Each PE core perform a Winograd convolution operation and calculate the output memory address based on the current $[ID]$ and $[OD]$. The compute near memory will add up convolution results for tiles with the same $[OD]$ to obtain the final output feature map, and send the result to the output SRAM.

3.1 PE Design

The PE performs is provided with U and V matrix either from the controllers or the previous PE. The A transformation matrix is stored inside the PE. With an input signal denoting the size of the desired convolution, we will select parts of the A matrix according to **Figure 2**. We have added pipeline registers inside the PE for the intermediate steps to reduce the critical path. We can obtain the output matrix Y following **Equation 1**, and the pipeline is shown in **Figure 4**.

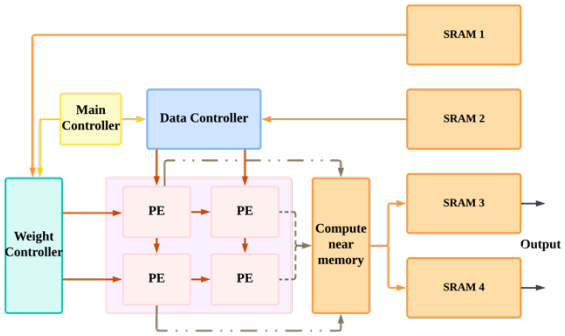

Figure 3: Block diagram of the Winograd systolic array architecture

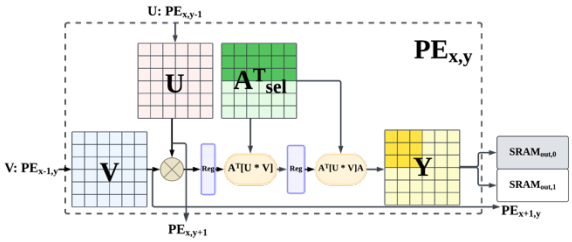

Figure 4: Block diagram of the WinoPE

3.2 Controller Design

There are three controllers in the design as shown in **Figure 5**. The main controller holds the information of sizes of all arrays in the calculation process. It is also responsible for counting the current $[ID]$ and $[OD]$. It monitors the calculation process and sends instructions to the other two controllers to generate correct data tiles and weight tiles.

The data controller receives signals from the main controller. It first calculates the address of required data and requests the data from memory. Then it performs two matrix multiplications done by shifting and adding (and subtracting) to produce matrix U for PEs. The data controller generates two tiles with different addresses which are sent to the two columns of PEs separately.

The weight controller has the similar functionality. It is used to prepare the weight tiles with two different $[OD]$ for the two rows of PEs. Since weight tiles are pre-computed and stored in memory, the weight controller simply fetches

and passes those to PEs without any processing. Notice that when there is only one output channel, the second row of PEs will not be utilized.

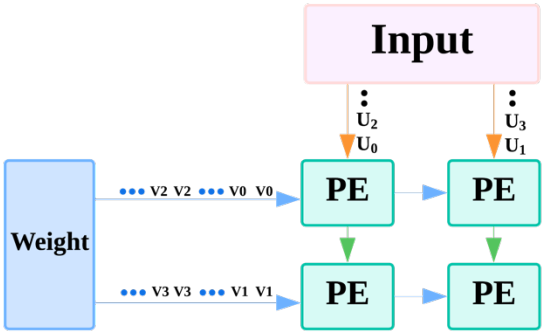

Figure 5: Input and weight data sequence from the controllers to a 4*4 PE array

3.3 Input memory design

We design the same controller for both the input SRAM and weight SRAM, implemented in the ***data_mem_top*** module. Both the input SRAM has the size of 512×128.

Before we start the calculation, we first scan all the data into the SRAM. Then the accelerator will start the computing step. In this step, the controller will send a package including the read address and valid bit to the memory controller, and the corresponding data will be loaded out at the next clock edge. We choose to use a dual-port SRAM because in every cycle, both the data controller and weight controller will send two requests to the memory.

3.4 Output memory and Computer-near-memory Design

In each cycle, the PE array will give out four result packages including address, data and valid bit. The output memory and computer near memory module will first read the address from the PE packages, then load the corresponding data in the SRAM. The compute near memory block will add up the data from the PE and memory and store the sum back to the memory.

Since the SRAM can only handle one load or write request in a single cycle, we decided to use a clock two times faster than the main clock, and we call it ***mem_clk*** in our design. We will load data from the memory when the main clock is high and write to the memory when the main clock is low.

We design four compute-near-memory blocks and two dual-port SRAM with size 512×128 to meet the computational requirements. This means we have a total four SRAM ports and four separate computer-near-memory blocks. One SRAM port and one compute-near-memory block can handle one PE package.

To efficiently control the mode of the SRAM, we designed a finite state machine with four stages. The four stages are scan in, load, write and scan out. After all calculations are finished, data stored in the memory is scanned and wrote into a text file.

4 Results

4.1 Validation Results

We have implemented a fully functional fixed-point model of our RTL design in MATLAB. The MATLAB model is used to generate text files for inputs to the Verilog testbench and output text files to verify correctness. Steps on running the testbenches and reproducing the results below are written in the README file.

4.1.1 MATLAB Fixed Point Validation

We used the MATLAB fixed-point model to test out different fixed-point representations to ensure a tolerable precision and overflow. We used the MATLAB script as a ground truth to compare our PE and memory output data.

4.1.2 Verilog PE Testbench

We have verified a single PE is able to produce correct results for both 1×1 and 3×3 filter size. The MATLAB result and the PE testbench matches up 100%. Further tests with SRAM, data controllers are performed, and results are shown in the next two sections.

4.1.3 Verilog [ID] = [OD] = 1 Testbench

To test all the modules working together, we used MATLAB to generate the input matrix, and the V matrices in hexadecimal. These numbers are loaded into the SRAM module. The output is written to the text file where we checked the difference with our ground truth MATLAB fixed point model. It has been shown that our Verilog testbench result matches the MATLAB output. **Figure 6** compares the Winograd systolic array result with an ideal floating-point model. The maximum error between floating-point convolution and fixed-point PE output is 0.0532.

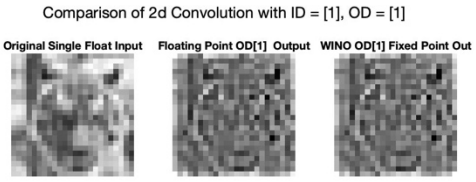

Figure 6: Comparison of floating-point model with WinoPE output for $K = 3, ID = OD = 1$

4.1.4 Verilog [ID] = 2, [OD] = 4 Testbench

To test a more complex convolution case, we followed steps above to produce data for [ID] = 2 [OD] = 4. From **Figure 7**, we can see the 4 output feature maps and the comparison with the ideal floating-point output. It has been shown that our Verilog testbench result matches the MATLAB output.

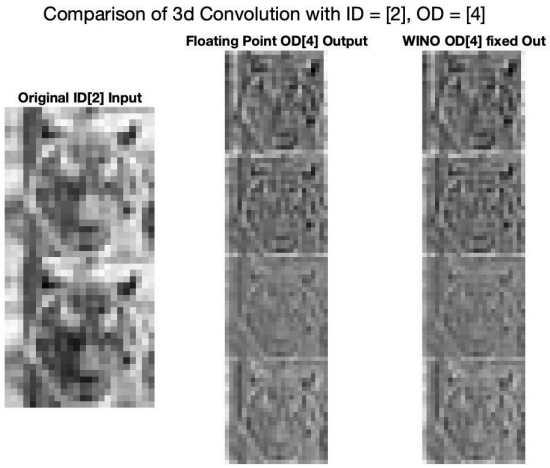

Figure 7: Comparison of floating-point model with WinoPE output for $K = 3, ID = 2\ OD = 4$

4.2 Synthesis results

The total area of our design is $0.598\ mm^2$. The clock period is $12\ ns$ and the slack is met in all timing reports.

According to the power estimation report, the total power of our accelerator is $0.282\ W$. The memory power is $0.242\ W$, which made up 85.96% of the total power. The combinational logic power is 0.036W, which made up 13.04% of the total power.

4.3 Analysis of results

The clock period in this design is 12ns and the maximum clock period in the critical path is 12. Considering that this design still has much potential in optimization, like designing a better architecture for matrix multiplication or enabling a second clock for the interface with memory, this result is acceptable.

The area of this design is not large considering the existence of the SRAM. In our design, we are only using four PEs for calculation. If we increase the scale to 4*4, the area grown is estimated to be less than four times the current status while the performance is four times better. There will also be more optimization potential if the scale is larger.

The power cost in this design mainly comes from the SRAM. Apart from the memory, combinational logic occupies most of the power cost. Since we support multiple kernel sizes in this design, some part of logic may be unused in certain situations. Power gating may be applied to those logic to reduce the power usage.

5 Future Works

We would like to optimize our design to be more area and power efficient. This can be done by decreasing the throughput of the PE so there would be less stress on the input and output bandwidth.

Furthermore, we propose a new controller and PE arrangement with an improved data output sequence. This design assumes the $[OD]$ is divisible by 4 (true for most state-of-the-art CNNs), and the new data flow ensures that all $[OD]$ of an input tile are calculated. This enables the ability to start the next convolution layer without the need to wait for the whole 3d convolution operation to finish. **Figure 8** provides an example of the when $[OD] = 8$. The black texts represent data corresponding to the first tile, while the red texts represent data corresponding to the second tile.

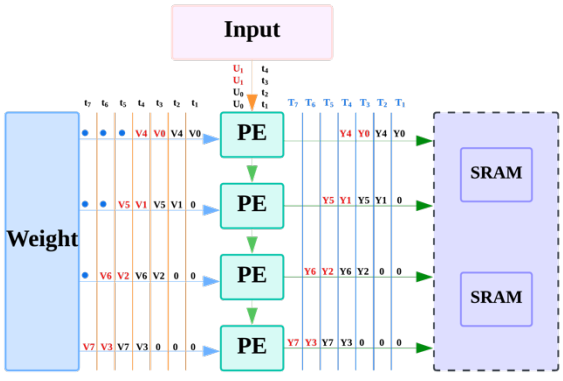

Figure 8: Data flow of the proposed controller

6 Conclusion

We have successfully implemented a Winograd systolic array that supports multiple kernel sizes. We have validated our RTL outputs against our MATLAB fixed-point model and compared it with the ground truth floating point results. Our design is the size of $0.598\ mm^2$ with a clock period is $12\ ns$ and consumes $0.282\ W$. The memory consumes 85.96% of the total power while the combinational logic power consumes13.04%. The results are visualized in MATLAB and our fixed-point error is within the tolerable range.

While the reference paper is helpful, it fails to disclose the design of their memory and data flow. This led to memory bandwidth challenges where we have to make the trade-off between power and throughput. We have proposed a set of improvements and a new architecture to optimize for these short-comings.

7 Contributions

Each team member contributed to the design, implementation, and testing phases of the project as follows:

Wenjie Geng: Led the architecture design. worked on RTL implementation of the controllers and memory systems, created testbenches, debugged the Verilog modules, and contributed to synthesis efforts. The architecture is inspired by the reference paper [2] and our controller and memory design are original ideas and implemented from scratch.

Taoran Ji: Led module integration and testing. Contributed to the RTL implementation of the PE and controller modules, created testbenches, debugged the Verilog modules, and contributed to synthesis efforts. The PE architecture is the same as the design reference paper [2], while we added pipeline register to reduce the critical path.

Jason Ning: Led the paper research process. Developed the fixed-point MATLAB model, contributed to the of the PE pipelining and Verilog debugging efforts, and generated text files for testbench input and output comparison. The MATLAB models are written from scratch with the same PE implementation as reference paper [2].

Yuxi Peng: Assisted RTL memory design and created figures and diagrams for the report. The architecture diagrams in the report are inspired by reference paper [2] and revisions are made to reflect our design changes in the project.

8 References

[1] A. Lavin and S. Gray, "Fast Algorithms for Convolutional Neural Networks," *arXiv:1509.09308 [cs]*, Nov. 2015, Accessed: Nov. 30, 2022. [Online]. Available: https://arxiv.org/abs/1509.09308

[2] X. Liu, Y. Chen, C. Hao, A. Dhar, and D. Chen, "WinoCNN: Kernel Sharing Winograd Systolic Array for Efficient Convolutional Neural Network Acceleration on FPGAs," *arXiv.org*, 2021. https://arxiv.org/abs/2107.04244

[3] S. Winograd, *Arithmetic Complexity of Computations*. SIAM, 1980.

[4] Abhinav Podili, C. Zhang, and V. K. Prasanna, "Fast and efficient implementation of Convolutional Neural Networks on FPGA," Jul. 2017, doi: https://doi.org/10.1109/asap.2017.7995253.

[5] S. Kala, J. Mathew, B. R. Jose, and S. Nalesh, "UniWiG: Unified Winograd-GEMM Architecture for Accelerating CNN on FPGAs," Jan. 2019, doi: https://doi.org/10.1109/vlsid.2019.00055.