Hardware Implementation of 2D Convolution

A Time Efficient Architecture for Image Processing

VLSI Signal Processing
AVM867

Date: April 24, 2025

Indian Institute of Space Science and Technology
Trivandrum

Submitted By:
Sanyam Gupta (SC22B114)
Jyothin Sai Swaraj (SC22B099)
Aryan Gujral (SC22B170)

Submitted To:
Dr. Sheeba Rani J
Professor
Avionics Department

Table of Contents

Contents

1 Abstract

This project focuses on the hardware implementation of 2D convolution on FPGA for image processing. The design uses a systolic array architecture to achieve high computational efficiency using techniques like pipelining and parallelism. The dual-window architecture,accessing two moving windows in parallel to process the image, demonstrates twice the throughput of the conventional single-window approach while consuming low area. FIFO buffers, implemented with dual-port FPGA RAM, are employed to handle these sliding window operations. The multiplication operation is optimized using LUT-based constant coefficient multipliers to minimize silicon area and power consumption. Additionally, an adder tree architecture comprising of Kogge-Stone Parallel Prefix Adder, is used for parallel grouping and addition, significantly reducing computation time and enabling seamless pipelined operations.

2 Introduction

High computing power is needed for modern image processing and computer vision techniques, particularly when real-time analysis of high-resolution images is required. The two-dimensional (2-D) convolution is essential to these applications (such as image filtering, picture restoration, feature detection, object tracking, template matching, etc.). Consequently, there is a lot of interest in the design of effective convolvers for both military and commercial applications. Although it may be simpler, the aforementioned applications are not as fast when implemented on a general-purpose computer. The extra restrictions placed on memory and other peripheral device management are the cause. Hardware tailored to a particular application provides significantly faster performance than software. The last few years have seen an unprecedented effort by researchers in the field of image processing based 2D convolution using FPGA, and this report deals with one of such architectures based on 3x3 kernel operation on an image of 512x512 pixels. The aim of the work proposed in this paper is to present a general architecture for the systolic array that can be used in implementing the operations of the two-dimensional convolution, employing high performance adder and multiplier architectures.

3 Literature Review

3.1 2D Convolution

2D-Convolution, is most important to modern image processing. The basic idea is that a window of some finite size and shape is scanned over an image. The output pixel value is the weighted sum of the input pixels within the window where the weights are the values of the filter assigned to every pixel of the window. The window with its weights is called the convolution mask. Mathematically, convolution on image can be represented by the following equation:

$$y(i,j) = \sum_{l=0}^{M-1} \sum_{k=0}^{N-1} h(k,l).x(i-k, j-l)$$

3.2 Moving Window Operation

The algorithms implemented in this work use the moving window operator. The moving window operator usually process one pixel of the image at a time, changing its value by some function of a local region of pixels (covered by the window), but this architecture involves processing of two pixels parallely. The operator moves over the image to process all the pixels in the image. Each pixel in the output image is produced by sliding an 3x4 window over the input image and computing an operation according to the input pixels under the window and the chosen window operator. It results in two pixel values which are assigned to the centre of the window in the output image, as shown below in Figure 1.

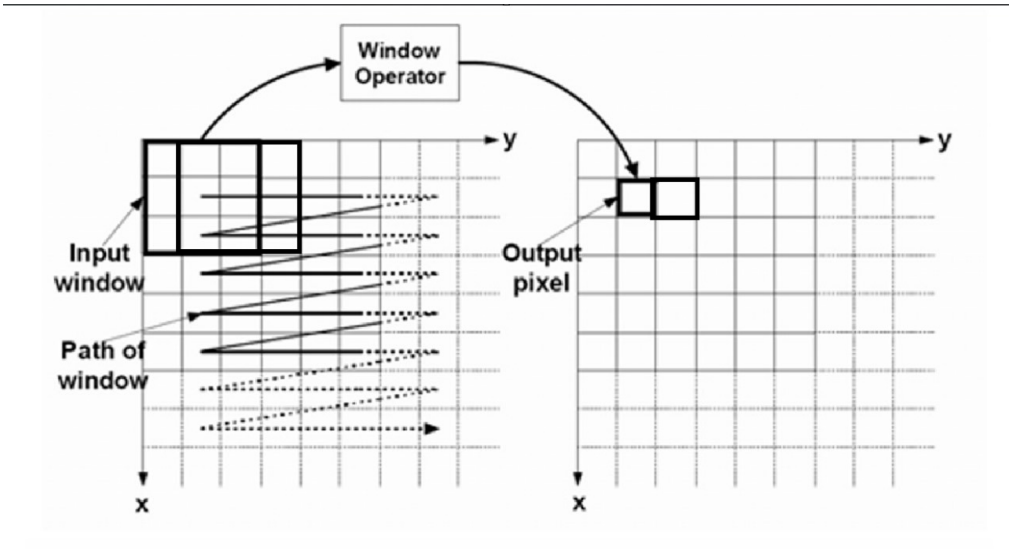

Figure 1: Window Operation

4 Convolution Hardware Implementation

An efficient architecture of high performance is required in the applications that use the 2D convolution to be processed in real time. For this purpose, a systolic array architecture is used in this work (Figure 2).

The algorithms implemented in this work use the moving window operator. The moving window operator processes two pixels of the image at a time, changing their value by some function of a local region of pixels (covered by the window). The operator moves over the image to process all the pixels in the image. A 3x4 moving window is considered as in Figure 2.

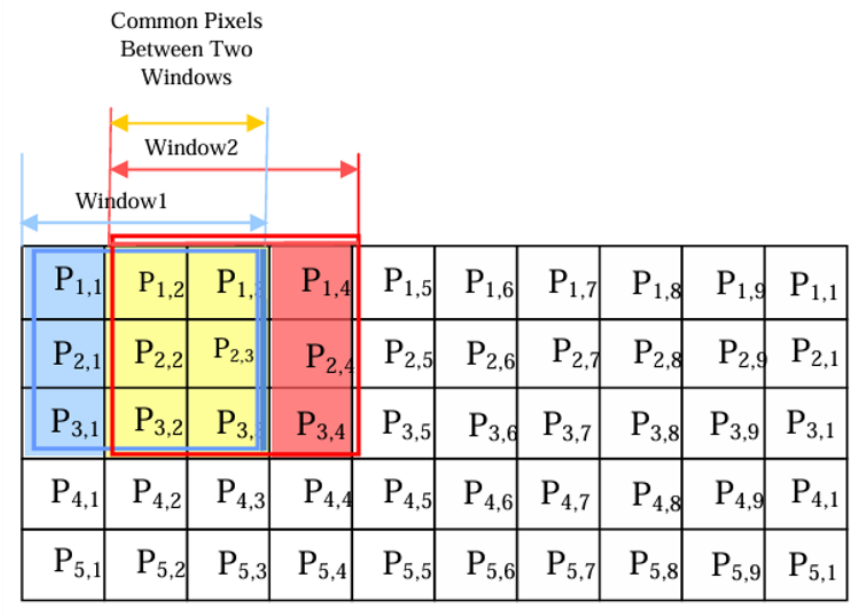

Figure 2: Dual Window Operation

This is achieved by sharing two adjacent pixels the same memory address. Thus, 16-bit memory width is used for this purpose. The idea behind this type of processing is extracted from the fact that there is a common pixels between two adjacent moving windows as shown in Figure 2.

The first pixel should be stored in the least significant byte(LSB) of the first memory location, and the 2nd pixel in the MSB and so on for all the rest of pixels. From Figure 2, one can see that there are six common pixels between two adjacent

windows. Therefore, the data of two 3x3 windows can be available by just reading one window of 3x4. The architecture of this technique is shown in Figure 3.

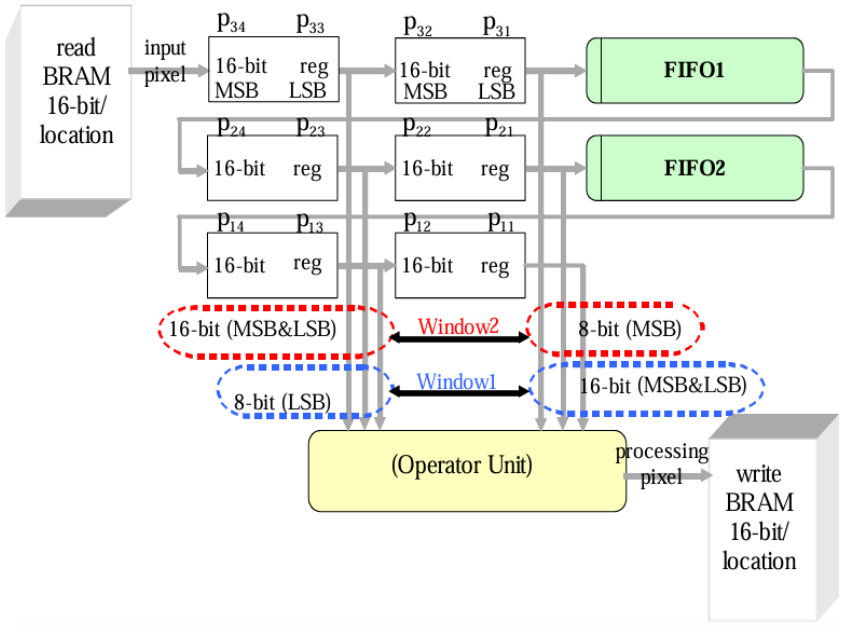

Figure 3: Architecture

The advantage of using this technique is that the operation required to process one image can be duplicated at the same time duration. The HDL implementation is shown below.

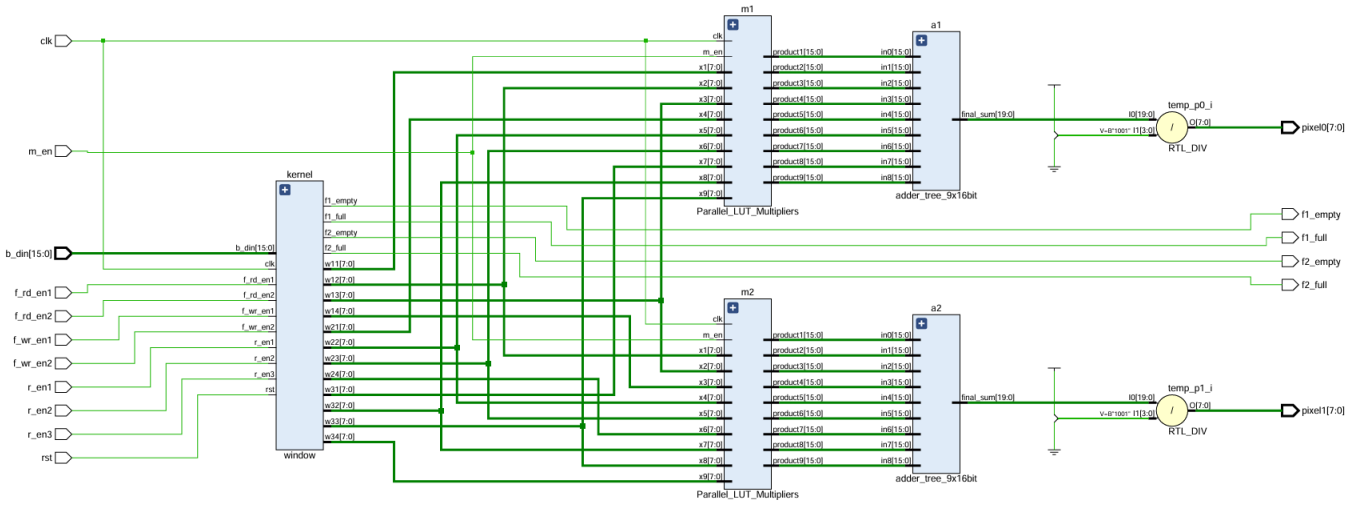

Figure 4: Designed Hardware Schematic

Each of the blocks involved have been explained in detail, hereafter.

4.1 The Image Memory

The gray-scale image is converted into a *.coe* file containing rows of 2 pixels clubbed together and those 16-bit words arragned in reverse order i.e., from bottom right corener of image to top left corner traversed row wise, using a python code.

A block RAM of depth 2^{17}, initialized using IP blocks, is used to store those values and supply it to kernel window architecture

when turned on. Similarly a BRAM at output is used to fetch the resultant 16-bit words (2-pixels); and subsequently an image is generated using a python code.

4.2 The Kernel Block

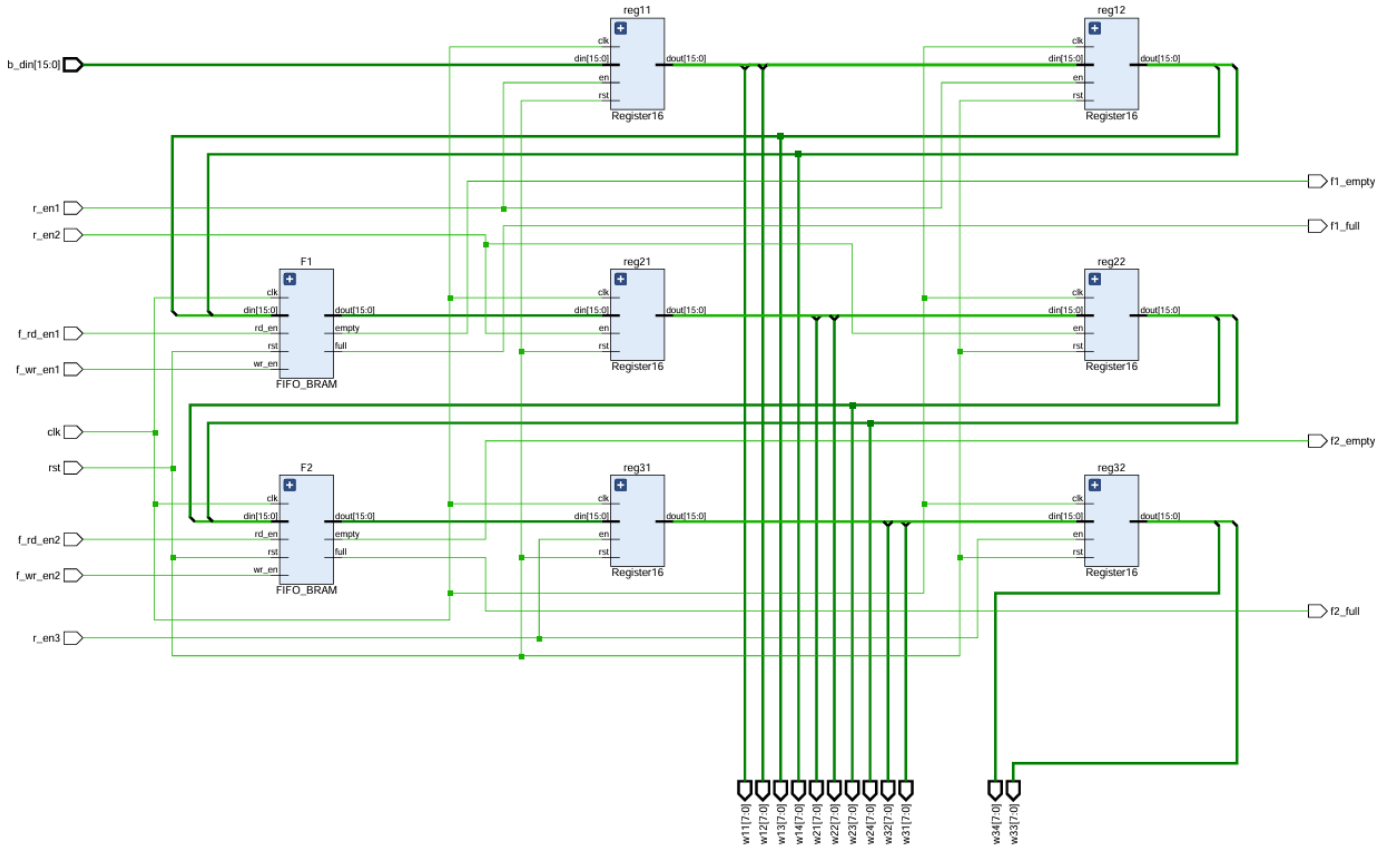

Figure 5: The Window Architecture

If real-time processing of the video stream is required, N x M pixel values are needed to perform the calculations each time the window is moved and each pixel in the image is read up to N x M times. Memory bandwidth constraints make obtaining all these pixels each clock cycle impossible unless some form of local caching is performed. Input data from the previous (N-1) rows can be cached using a shift register (or FIFO buffer) for when the window is scanned along subsequent lines. This leads to the block diagram shown in Figure 5.

Pipelining, parallelism and systolic techniques are used to make the computation performs in high efficiency. For the pipelined implementation of image processing algorithms all the pixels in the moving window operator must be accessed at the same time for every clock. The First In First Out(FIFO) buffers are used to create the effect of moving an entire window of pixels through the memory for every clock cycle. A FIFO consists of a block of memory and a controller that manages the traffic of data to and from the FIFO. The FIFO's are implemented using circular buffers constructed from multiport block RAM with an index keeping track of the front item in the buffer. This allows a throughput of one pixel per clock cycle.

For a 3x4 moving window two FIFO buffers are used. For every clock cycle, a pixel is read from the RAM and placed into the top left corner location of the window (the *b_din* 16-bit input port).

The 8-bit pixels are suitably tapped from the 16-bit registers; pusing out 12 pixels for dual-window opeartion into the Multipliers followed by Adders as shown in Figure - 4. There are various control signals involved, which help out in pipeling operations.

4.3 The Multiplier

When convolution on 8-bit pixels is required, each multiplier performs one 8×8 pixel-weight product. In this system we intended to design the convolution engine to be as a part of larger system. In order to minimize the area cost and the total power consumption and also to simplify the implementations, Look up tables (LUTs) have been used in the multiplication operations. A constant coefficient multiplier is built in by using ROMs memories to parallelize the read operation and to reduce the size of the LUT, and consequently reducing the size of the silicon area. In principle, the evaluation of any finite function can be carried out using a look-up table (LUT) memory that is addressed with the argument for the evaluation and whose output is the result of the evaluation. Unfortunately, the use of a single LUT for the multiplication is unlikely to be practical for any but the smallest argument, because the table size grows rapidly with the width of the argument. Therefore the solution is to split the argument, use LUTs, and then use a tree of adders. An example of this is given in Figure 6.

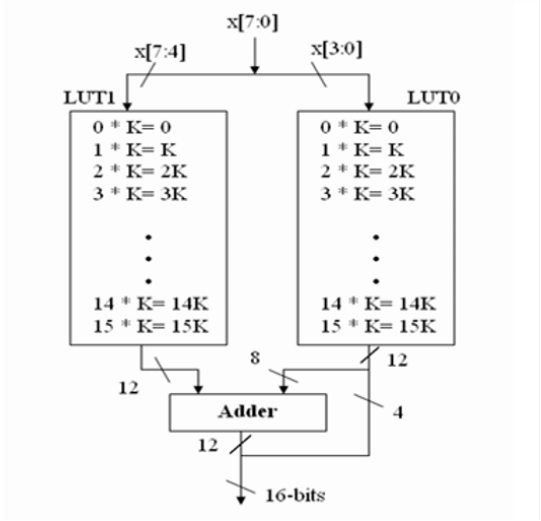

Figure 6: LUT Based Multiplier Logic

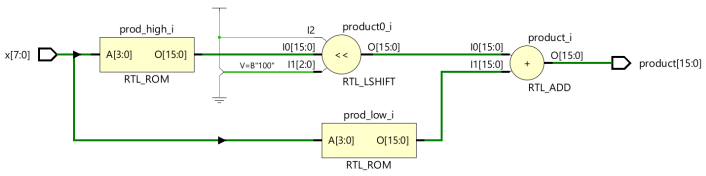

Figure 7: The LUT Based Multiplier

The nine 16-bit independent multiplications obtained in this way, are then added by the Adder Tree, which generates a 20-bit result (without loosing the precision).

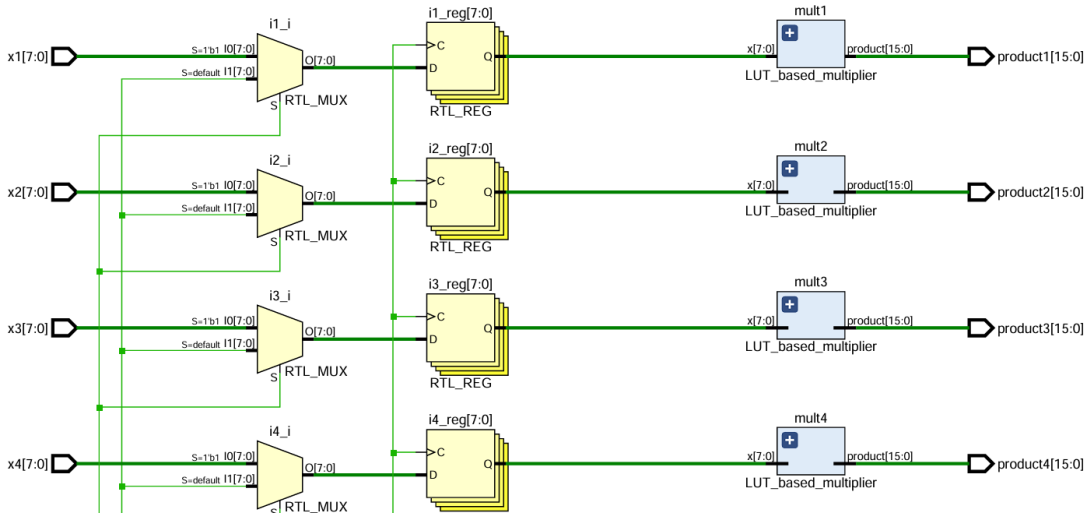

Figure 8: The Block Multiplier - 9 Parallel Multiplications

Two such multipliers are used in parallel as clear fromshown in Figure 4.

4.4 The Adder

Conventional adders like the ripple carry adder (RCA) are simple but suffer from long propagation delays. To overcome this, **parallel prefix adders** such as the **Kogge-Stone Adder (KSA)** are introduced. KSA provides superior performance for wide adders due to its **logarithmic delay** and minimal fan-out, making it highly suitable for high-speed arithmetic.

The aim is to develop an efficient architecture to sum nine 16-bit inputs , resulting from the multiplier in the previous section, using a structured and modular **parallel prefix adder tree** based on KSA. This is particularly suited for applications like image filtering, where pixel data are naturally positive .

4.4.1 Kogge-Stone Adder

Bitwise Generate and Propagate

For each bit i:

$$G_i = A_i \cdot B_i \quad \text{(Generate)}$$

$$P_i = A_i \oplus B_i \quad \text{(Propagate)}$$

Recursive Group Logic

$$G_{i:j} = G_i + (P_i \cdot G_{i-1}) + \cdots + (P_i \cdot \cdots \cdot P_{j+1} \cdot G_j)$$

$$P_{i:j} = P_i \cdot P_{i-1} \cdot \cdots \cdot P_j$$

Prefix Tree Computation

For $i > j$:

$$G_{i,j} = G_{i,k} + P_{i,k} \cdot G_{k-1,j}$$

$$P_{i,j} = P_{i,k} \cdot P_{k-1,j}$$

The tree depth is $\log_2 n$, which governs the carry propagation time.

4.4.2 Multi-Stage Parallel Adder Tree

To sum 9 unsigned 16-bit inputs:

- **Stage 1:** 4 KSAs (16-bit input) to compute sums of pairs.
- **Stage 2:** 2 KSAs (17-bit input) to sum intermediate results.
- **Stage 3:** A KSA (18-bit input) to sum previous two results.
- **Stage 4:** A KSA (19-bit input) to add the 9^{th} input (zero-extended to 19 bits).

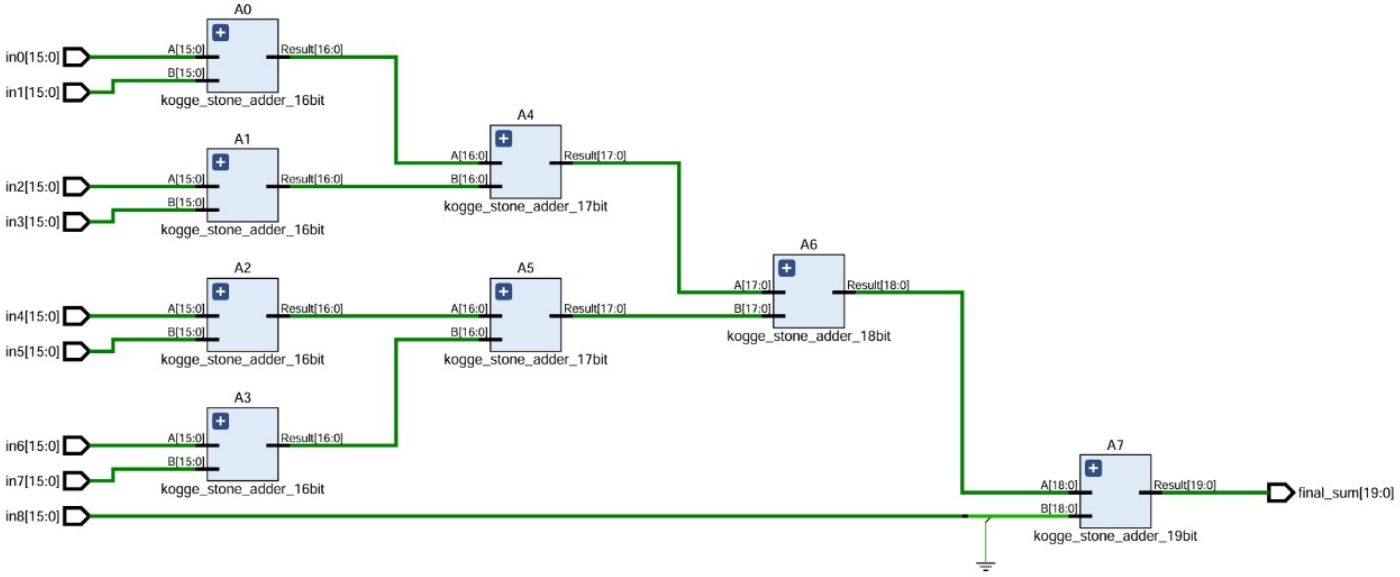

Figure 9:

This project demonstrates a high-speed, modular addition system using Kogge-Stone Adders. By designing 16, 17, 18, and 19-bit KSA modules and integrating them into a 4-stage tree, we achieve fast summation of 9 inputs with minimal delay.

5 Performance and Complexity

The design architecture is modeled by using Verilog HDL language and implement on Zynq UltraScale+ MPSoCs FPGA board.

5.1 Convolver

As per STA, the Critical Path Delay is apprximately 1.512 ns, with a clock period constraint set to 10 ns.

Since there are 2 output pixels every clock cylce, once the hardware is fully activated, the throughput is roughy **200 million pixels per second**.

5.2 Adder

KSA has a Time Complexity of $O(\log_2 n) + O(1)$ and Minimal fan-out (each signal drives fixed destinations). KSA offers a more regular structure and minimal fan-out, giving faster paths and better scalability in hardware. KSA also results in regular, predictable wiring, which simplifies routing and reduces layout congestion.

- Logarithmic carry computation
- Low fan-out and uniform structure
- Hardware efficiency for image processing

6 Conclusion

It is found that the throughput of the proposed architectures become twice the throughput of the conventional architecture which uses single window. Building on this; one can conclude that the architecture can be used in very high speed and video applications, in real time. Using LUTs based multiplication leads to further improvement on the overall throughput, and the system speed consequently increases. The results are sown in following images; after applying a Box Blur(Averaging) filter kernel.

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

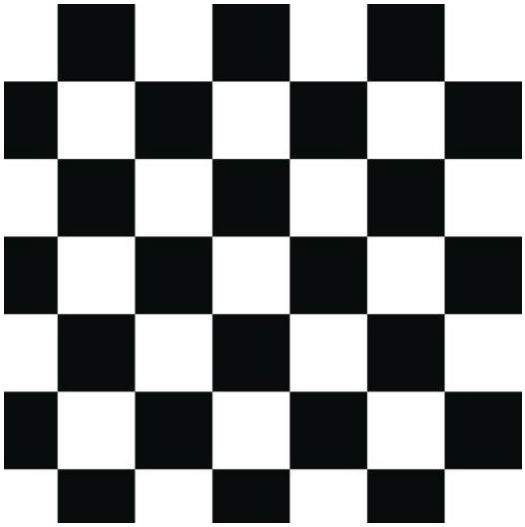

Figure 10: Original

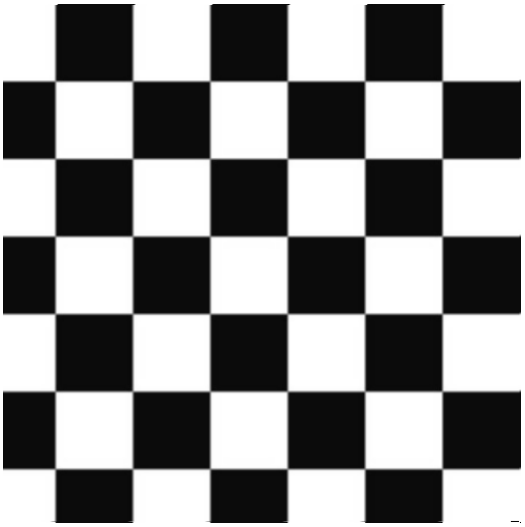

Figure 11: Blurred

Figure 12: Original

Figure 13: Blurred

7 References

- Hardware Implementation of 2D convolution on FPGA Shefa A. Dawwd, Faris S. Fathi
- Constant Coefficient Multiplication in FPGA Structures, Kazimierz Wiatr, Ernest Jamro, IEEE
- Implementation of a Parallel Prefix Adder Based on Kogge-Stone Tree Yancang Chen, Minlei Zhang, Pei Wei, Sai Sui, Yaxin Zhao and Lunguo Xie