Systolic Array Architecture based Image Binarization using Basys3 FPGA Board

Chittela Akhila, SR.No.25340, M.Tech(Res.), Electronic Systems Engineering, DESE, akhilac@iisc.ac.in
Jangam Abhinav, SR.No.24925, M.Tech, Instrumentation Systems, IAP, abhinavjanga@iisc.ac.in
Indian Institute of Science, Bangalore-560012

***Abstract*—This report presents an implementation of image binarization using a systolic array architecture deployed on a Basys3 FPGA board. The binarization process, using DNN concept which involves matrix operations such as thresholding, is performed on hardware using a custom-designed systolic array and interfaced via the UART protocol. The system efficiently transfers image data between a host computer and the FPGA, where the core computations are offloaded. The results obtained show a comparison between software-based matrix operations and the FPGA-based approach.**

***Index Terms*—Systolic array, Basys3, Image binarization, DNN, UART.**

I. Introduction and Motivation

The Deep Neural Networks (DNNs) have become a cornerstone of modern computer vision tasks, including image classification, segmentation, and enhancement. At their core, DNNs rely heavily on dense matrix operations, particularly in layers involving convolutions and fully connected computations. These operations typically involve multiplying large $N \times N$ matrices, which have a computational complexity of $\mathcal{O}(N^3)$ on conventional von Neumann architectures. This poses significant challenges in terms of processing speed and energy efficiency.

To address this, hardware-accelerated architectures such as systolic arrays offer a promising solution. A systolic array (SA) processes data in a pipelined fashion through a grid of processing elements (PEs), enabling parallel computation with significantly reduced latency. For many matrix operations, systolic arrays can achieve a time complexity of $\mathcal{O}(N)$, making them highly efficient for DNN workloads.

In this work, we have explored a systolic array-based architecture implemented on the Basys3 FPGA board to perform image binarization, which is a fundamental image preprocessing step.

II. Background Study

FPGAs, such as the Basys3 with the Artix-7 FPGA, provide a flexible platform for implementing custom systolic architectures while optimizing power and performance trade-offs.

A. Systolic Array Architecture

Systolic array architecture is a specialized hardware design used to efficiently perform matrix operations, particularly matrix multiplication. It comprises a grid of interconnected

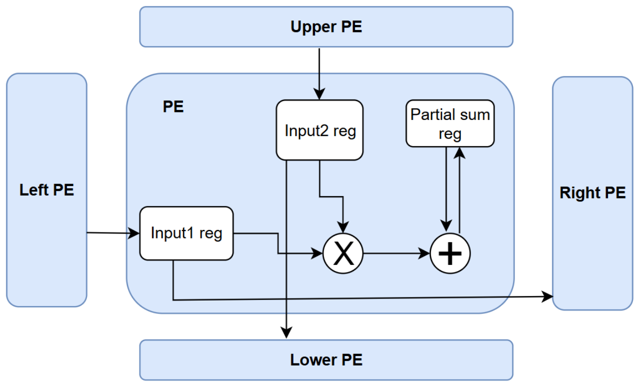
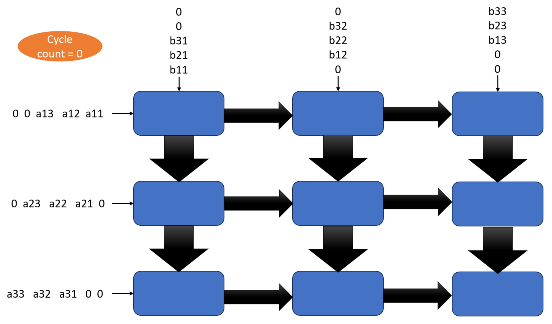
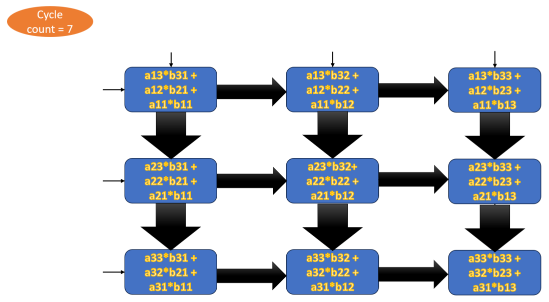


Fig. 1: Processing Element (PE) of Systolic Array



(a). Initial SA architecture when cycle count is zero


Cycle count = 7
a13*b31 + a12*b21 + a11*b11
a13*b32 + a12*b22 + a11*b12
a13*b33 + a12*b23 + a11*b13
a23*b31 + a22*b21 + a21*b11
a23*b32+ a22*b22 + a21*b12
a23*b33 + a22*b23 + a21*b13
a33*b31 + a32*b21 + a31*b11
a33*b32 + a32*b22 + a31*b12
a33*b33 + a32*b23 + a31*b13

(b). Output matrix values after 7 cycles

Fig. 2: Systolic Array Architecture for 3x3 Matrix Multiplication

processing elements (PEs), also known as MAC (Multiply-Accumulate) units. The name *"array"* comes from the grid structure, while *"systolic"* reflects the rhythmic flow of data across the PEs — inspired by the pumping action of the heart. Fig. 1 shows the MAC operaction that happens in a PE.

For a $N \times N$ matrix multiplication it takes (2*N+1) cycles to obtain the output product matrix. A demonstartion of 3×3 matrix multiplication using systolic array architecture is shown in fig. 2.

B. UART Protocol

UART (Universal Asynchronous Receiver-Transmitter) is a simple, asynchronous serial communication protocol used for data exchange between digital systems. It is a One-to-one communication (point-to-point) and it uses two main wires whic are Tx (Transmit) and Rx (Receive). This simplicity and reliability make UART protocol suitable for FPGA-based communication.

In this project, UART facilitates data transfer between a host PC and the FPGA board, enabling the sending of image and trained weight data matrices to the hardware and receiving binarized results back. Typically, baud rates like 9600, 57600, 115200 bps are used. In this design, a high baud rate of 921.6 kbps is used. The overall system flow is shown as a block diagram in fig. 3.

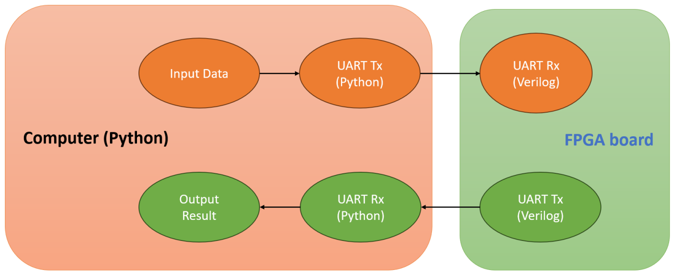


Fig. 3: Overall System Flow

C. Image Binarization

Image binarization is the process of converting a grayscale image into a binary image by applying a threshold. Pixels with intensity values above a certain threshold are assigned a value of 1 (white), and those below it are set to 0 (black). This process is widely used in computer vision and document analysis to reduce image complexity, preparing data for further tasks like OCR, segmentation, or edge detection.

III. Design Space Exploration and Design Strategies

Fig. 4 illustrates the complete data flow of matrix multiplication implemented on the Basys3 FPGA using a systolic array architecture. The pipeline begins with receiving the input matrices over UART and ends with the transmission of the computed output matrix.

A. Detailed Flow Description

1) UART Rx: The input matrices are received serially from a host device using the UART (Universal Asynchronous Receiver/Transmitter) protocol, which is suitable for low-speed, reliable FPGA-host communication.

2) Input BRAM: Data is stored in a dual-port Block RAM of depth $2 \times N \times N$ and width of 1 byte. This allows decoupling of communication and computation, enabling fast access and parallel read/write.

3) Input Buffer: The buffered input organizes the matrix data into a suitable layout for the systolic array. Buffers are preferred over FIFOs here due to their ability to provide random access and synchronized control, which is critical for matrix multiplication where row/column access patterns are non-linear.

4) Systolic Array Computation: The buffered data is fed into a systolic array, where processing elements perform multiply-accumulate operations in a pipelined manner. This structure allows matrix multiplication to be performed in $\mathcal{O}(N)$ time compared to $\mathcal{O}(N^3)$ in software.

5) Output Buffer: After computation, the output matrix is temporarily stored in a buffer of size $N \times N$ with 4-byte wide elements. This acts as a staging area before writing to memory and supports data verification.

6) Output BRAM: The results are stored in another BRAM of depth $N \times N$ and width of 4 bytes, allowing controlled access and preparing the data for transmission.

7) UART Tx: The processed data is sent back to the host system via UART, completing the data flow.

B. Significance of BRAM and Buffers

Block RAMs provide on-chip, fast, and reliable memory storage essential for real-time operations on FPGA. Buffers are used instead of FIFOs due to their flexibility in data access and better compatibility with the systolic array's pipelined structure. FIFO is inherently sequential, which limits its use in parallel matrix operations requiring multi-dimensional access.

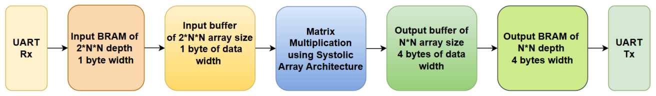

UART Rx
Input BRAM of 2*N*N depth 1 byte width
Input buffer of 2*N*N array size 1 byte of data width
Matrix Multiplication using Systolic Array Architecture
Output buffer of N*N array size 4 bytes of data width
Output BRAM of N*N depth 4 bytes width
UART Tx

Fig. 4: Block diagram of SA design onto the FPGA board

IV. Implementation Challenges

Several challenges are encountered during the implementation phase:

- **Timing Closure:** FSM transitions relying on immediate data availability from BRAM had to be deferred.
- **Resource Constraints:** FSM, control logic, and interconnects require a significant number of LUTs and Flip-Flops.
- **Debugging:** The FSM risked entering infinite loops if signals like done or buffer status flags were incorrectly timed or uninitialized, necessitating extensive use of simulation and the Integrated Logic Analyzer (ILA) for in-system debugging.

A. FSM Implementation Challenges

The finite state machine (FSM) governs the entire dataflow process, from input reception via UART to processing and output transmission. Several implementation challenges were addressed to ensure robust operation. The designed FSM is shown as in fig. 5.

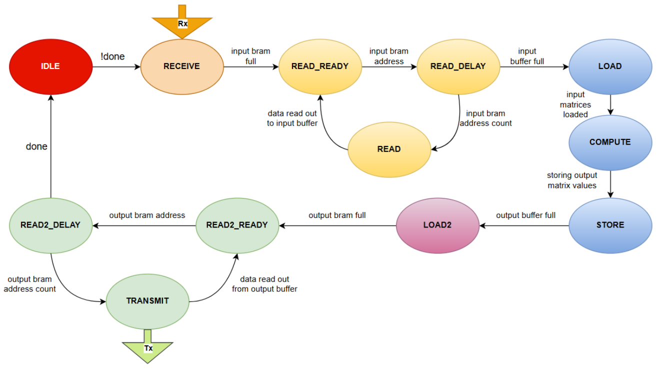


Fig. 5: FSM of a $N \times N$ Matrix Multiplication, implemented using Systolic Array and UART protocol

*1) **Handling BRAM Read Latency**:* Block RAM (BRAM) in FPGA exhibits a one-clock-cycle read latency. When an address is issued to BRAM, the corresponding data is only available in the subsequent clock cycle. To correctly capture the data without encountering timing issues, an extra state (`READ_DELAY` and `READ2_DELAY`) was introduced following the address issuance. This state allows the FSM to wait for one cycle before accessing the data, ensuring synchronization.

*2) **FSM Initialization and Deadlock Prevention**:* Without proper initialization signaling, the FSM risks entering a deadlock, particularly in the receive and transmit cycles. To counter this, a `done` flag was used. The FSM only transitions back to the `IDLE` state once all processing, including output transmission, is completed. This flag acts as a definitive indicator of computation completion, avoiding infinite loops.

B. Updated Processing Element for Image Binarization

The processing element (PE) was enhanced to support image binarization, extending beyond traditional matrix operations. The updated each new PE as depicted in fig. 6 now includes:

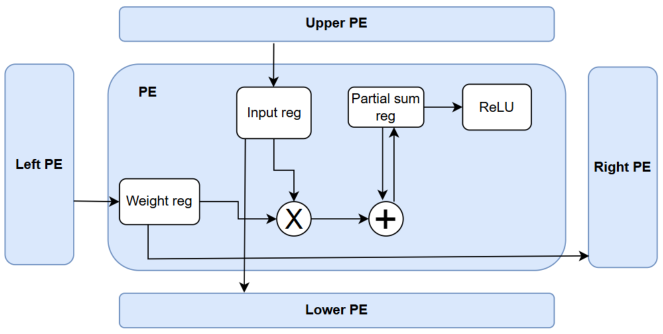


Fig. 6: Weight and ReLU included in updated PE

- **Weight Register:** Stores incoming weights from neighboring PEs.
- **Input Register:** Holds input image matrix values for computation.
- **MAC Unit:** Performs multiply-accumulate operations using input and weight registers.
- **Partial Sum Register:** Accumulates results from the MAC unit across cycles.
- **ReLU Activation:** Applies a non-linear function $\mathrm{ReLU}(x) = \max(0, x)$ to the accumulated output. This is crucial for tasks such as image binarization, classification, or inference in neural network models.

This modular design enables the PE to process both arithmetic and activation operations efficiently.

C. Output Stationary (OS) Weight Dataflow

The architecture adopts an Output Stationary (OS) weight dataflow model to support image binarization in the systolic array, which is as shown in fpg. 7. This dataflow strategy ensures that partial sums remain within the PE throughout the computation, significantly reducing memory access overhead.

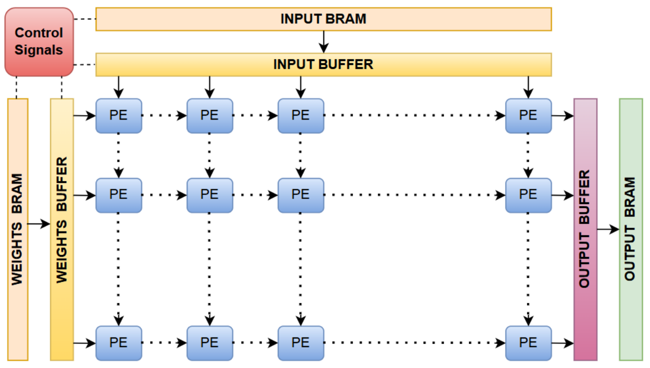


Fig. 7: OS Dataflow

1) Data Movement:

- **Weights:** Stream horizontally across PEs.
- **Inputs:** Stream vertically from top to bottom.
- **Partial Sums:** Stored and updated locally within each PE.

2) Advantages:

- **Reduced Memory Bandwidth:** Minimizes write-back and read cycles.
- **Improved Energy Efficiency:** Less external memory access is required.
- **High Throughput:** Each PE continues accumulating output without interruption, enhancing processing speed.

This dataflow model is ideal for FPGA implementations where on-chip memory and power efficiency are critical design parameters.

The combined FSM logic, PE enhancements, and OS dataflow strategy enable a scalable, efficient architecture suitable for real-time image processing.

V. Results

The final design achieved a clock frequency of **100 MHz** with successful processing of **8** $\times$ **8** pixel and **12** $\times$ **12** pixel matrices.

A. *Image Binarization Results*

The image binarization process using the proposed Systolic Array architecture on the Basys3 FPGA board was evaluated using different block sizes to assess the fidelity and accuracy of the hardware implementation compared to a software-based reference model. The comparison includes visual analysis of grayscale images, where hardware and software outputs are colorized for clarity.

Figure 8 shows the original input image, a yellow smiley face, which was resized and converted to grayscale before being processed. This preprocessing ensures that both the hardware and software operate on a consistent dataset.

Fig. 8: Input RGB Image

In Figure 9, results using an 8×8 pixel size are presented. The left image is the resized grayscale input; the middle image represents the software matrix output (colorized from white to blue), and the right image is the FPGA matrix output (colorized from white to red). Although the image retains the basic smiley face structure, the coarse resolution leads to a visible loss of detail. Differences between FPGA and software outputs are apparent, indicating possible effects from quantization.

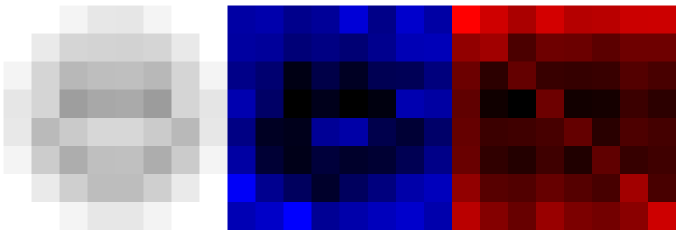


Fig. 9: Result comparison obtained using 8x8 pixel size

As shown in Figure 10, using a 12×12 pixel size significantly enhances image quality. The input image maintains better structure, and both FPGA and software outputs closely match. Slight mismatches may still be observed due to rounding effects, timing alignment, or fixed-point approximations within the FPGA pipeline. Overall, this resolution yields a visibly improved result with better fidelity to the original grayscale input.

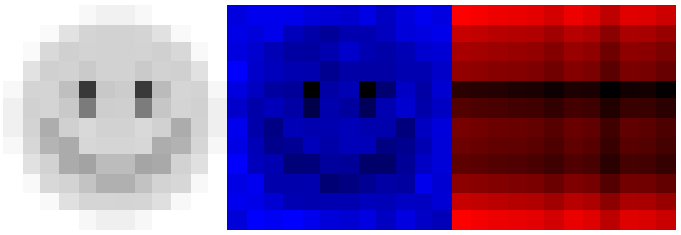


Fig. 10: Result comparison obtained using 12x12 pixel size

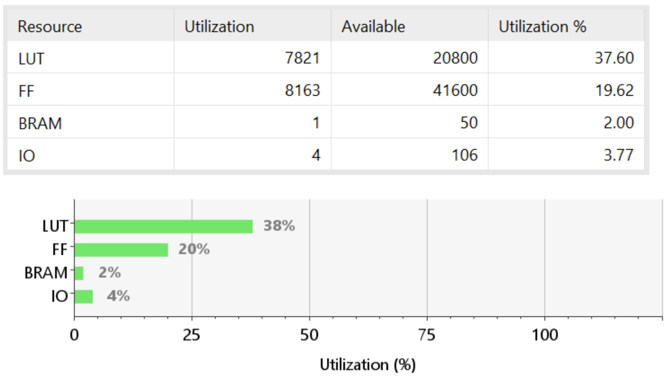
TABLE I: Output Image Result

Image	Color used	Meaning
Left	Grayscale (no colorize)	Original resized input
Middle	White $\rightarrow$ Blue	Software matrix output
Right	White $\rightarrow$ Red	FPGA matrix output

Table I describes the visual encoding used to distinguish between outputs. While the FPGA and software results are both single-channel grayscale outputs, colors are applied only for better visual comparison.

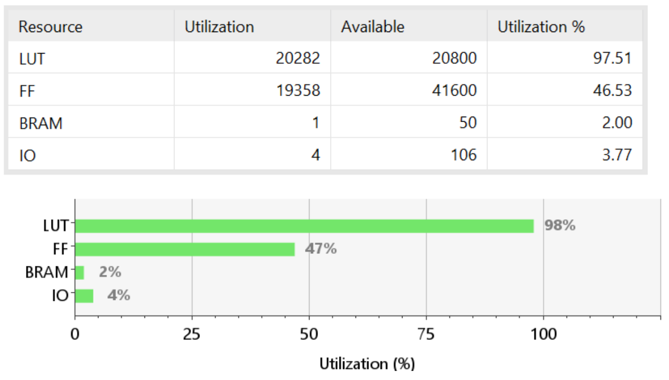
B. *Post-Route Results*

The resource utilization of 8×8 and 12×12 matrix multiplication using systolic array are as shown in fig. 11 and 12. Using a Basys 3 FPGA board, the maximum feasible implementation is a **12**$\times$**12** matrix multiplication, as it utilizes 98% of LUTs. Table II shows the slack and power metrics comparison.



Resource	Utilization	Available	Utilization %
LUT	7821	20800	37.60
FF	8163	41600	19.62
BRAM	1	50	2.00
IO	4	106	3.77

Fig. 11: Resource Utilization of 8×8 Matrix Muliplication



Resource	Utilization	Available	Utilization %
LUT	20282	20800	97.51
FF	19358	41600	46.53
BRAM	1	50	2.00
IO	4	106	3.77

Fig. 12: Resource Utilization of 12×12 Matrix Muliplication

TABLE II: Post-Route Results

Metrics	**8×8**	**12×12**
Slack	0.067 ns	0.137 ns
Power	0.103 W	0.133 W
Clock Frequency	100.67 MHz	101.39 MHz

VI. Conclusion

The systolic array design for image binarization using Basys3 FPGA board was successfully implemented and tested on the Basys3 FPGA board. The design met the performance goal for 8×8 pixel and 12×12 pixel inputs. Future work may involve scaling the design for larger matrices or avoiding the usage of buffers to design for large N values and deploying more advanced optimization techniques like dynamic frequency scaling.

References

[1] Kung, ”Why systolic architectures?,” in Computer, vol. 15, no. 1, pp. 37-46, Jan. 1982, doi: 10.1109/MC.1982.1653825.

[2] Xilinx, “Basys 3 FPGA Board Reference Manual,” Digilent Inc.

[3] M. Peemen, A. A. A. Setio, B. Mesman and H. Corporaal, ”Memory-centric accelerator design for Convolutional Neural Networks,” 2013 IEEE 31st International Conference on Computer Design (ICCD), Asheville, NC, USA, 2013, pp. 13-19, doi: 10.1109/ICCD.2013.6657019.

[4] Hao Lu, ”Design and implementation of a highly parallel systolic array on FPGA” *International Journal of Engineering Research and Management (IJERM)*, vol. 11, no. 3, pp. 41–45, Mar. 2024. [Online].

[5] Zhijie Yang, Lei Wang, Dong Ding, Xiangyu Zhang, Yu Deng, et al.. ”Systolic Array Based Accelerator and Algorithm Mapping for Deep Learning Algorithms”. 15th IFIP International Conference on Network and Parallel Computing (NPC), Nov 2018, Muroran, Japan. pp.153-158.

[6] Lym, Sangkug, and Mattan Erez. ”FlexSA: Flexible systolic array architecture for efficient pruned DNN model training.” arXiv preprint arXiv:2004.13027 (2020).

[7] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. ”Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks”. In Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA ’15). Association for Computing Machinery, New York, NY, USA, 161–170. AVilable on: https://doi.org/10.1145/2684746.2689060

[8] G. Yang et al., ”SA4: A Comprehensive Analysis and Optimization of Systolic Array Architecture for 4-bit Convolutions,” 2024 34th International Conference on Field-Programmable Logic and Applications (FPL), Torino, Italy, 2024, pp. 204-212, doi: 10.1109/FPL64840.2024.00036.

[9] Pong P. Chu, ”FPGA Prototyping by Verilog Examples: Xilinx Spartan-3 Version”. Hoboken, NJ: Wiley-Interscience, 2008.