Department of Electronic & Telecommunication Engineering
University of Moratuwa

EN4021 - Advanced Digital Systems

SIMD processor array for convolution neural network

Amarathunga D. N.	210037G
Jayathilaka D. E. U.	210254T
Pasira I. P. M.	210446J
Rajapaksha S. D. D. Z.	210508D

Submitted in partial fulfilment of the requirements for the module
EN4021 - Advanced Digital Systems
18th December 2025

1 Micro - Architecture

1.1 Instruction Set Architecture (ISA)

Table 1: Proposed SIMD CNN Accelerator Instruction Set Format

Instruction	**Opcode**	**Sub-Opcode**	**Image Addr**	**Image Size**	**Kernel Addr**	**Kernel Size**	**Stride**	**Output Addr**	**Padding**
NOP	00	00	0	0	0	0	0	0	0
CONV	00	01	10 bits	8 bits	10 bits	3 bits	2 bits	10 bits	1 bit
MAXPOOL	01	00	10 bits	8 bits	x	x	2 bits	10 bits	x
MINPOOL	01	01	10 bits	8 bits	x	x	2 bits	10 bits	x
AVGPOOL	01	10	10 bits	8 bits	x	x	2 bits	10 bits	x
RELU	10	00	10 bits	8 bits	x	x	x	10 bits	x
SIGMOID	10	01	10 bits	8 bits	x	x	x	10 bits	x
SOFTMAX	10	10	10 bits	8 bits	x	x	x	10 bits	x

A custom 48-bit Instruction Set Architecture (ISA) is proposed to control the SIMD processor array designed for accelerating convolutional neural networks on an FPGA-based SoC. The ISA follows a macro-instruction paradigm, where each instruction corresponds to a complete CNN operation such as convolution, pooling, or activation, rather than low-level arithmetic operations. This abstraction simplifies software control and allows the ARM processor to configure and invoke CNN layers efficiently.

Each 48-bit instruction encodes the operation type (opcode and sub-opcode), local buffer addresses for input, kernel, and output data, and key layer parameters including image size, kernel size (1×1 to 7×7), stride, and optional padding. The ISA supports convolution, max/min/average pooling, and activation functions including ReLU, Sigmoid, and Softmax, thereby addressing the core computational requirements of modern CNNs. Overall, the proposed ISA provides a compact yet flexible control interface that enables efficient coordination between the ARM processor, DMA engine, and SIMD accelerator while maintaining low software complexity and high hardware utilisation.

For the purpose of the project demonstration, a simplified version (34 bit) of the proposed Instruction Set Architecture was implemented. This reduced ISA supports the core CNN operations required for the demo, while maintaining a compact instruction format. The simplified ISA enabled faster implementation and verification of the SIMD processor array.

Table 2: Simplified Instruction Set Architecture Used for Project Demonstration

Instruction	Opcode (2)	SubOpcode (1)	Image Addr (10)	Image Size (6)	Kernel Size (3)	Output Addr (10)	Stride (2)
NOP	00	0	0	0	0	0	0
CONV	00	1	10 bits	8 bits	10 bits	10 bits	2 bits
MAXPOOL	01	0	10 bits	8 bits	x	10 bits	2 bits
RELU	10	0	10 bits	8 bits	x	10 bits	x
SIGMOID	10	1	10 bits	8 bits	x	10 bits	x

1.2 Overall System Micro Architecture

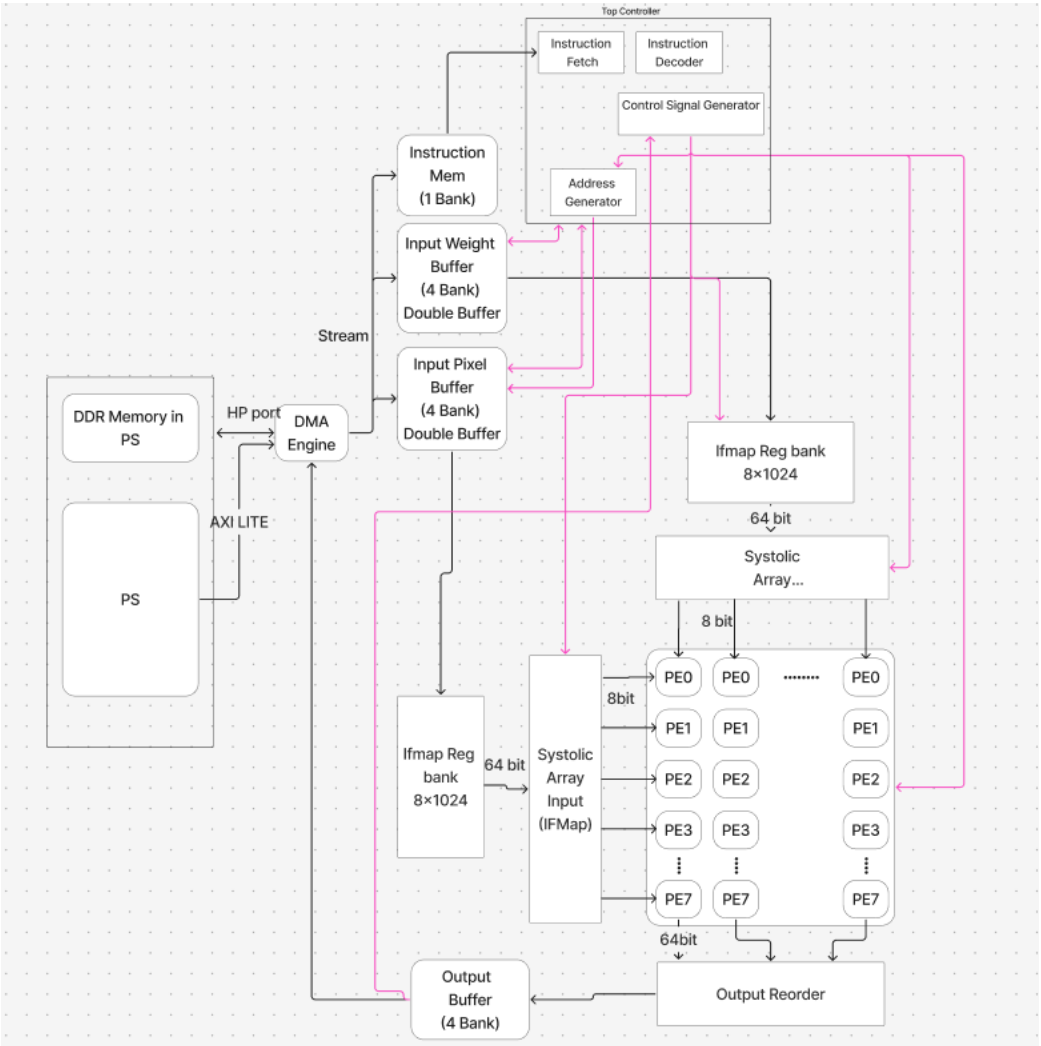

Figure 1: Top-level architecture of the SIMD processor array integrated into an FPGA-based SoC

The proposed architecture implements a SIMD processor array as a hardware accelerator within an FPGA-based system-on-chip. The ARM Processing System (PS) is responsible for high-level control, including instruction programming, DMA configuration, and overall execution sequencing. Input feature maps and kernel weights are stored in DDR memory and streamed into the programmable logic (PL) using a DMA engine through high-performance AXI interfaces.

Within the programmable logic, streamed input pixels and weights are buffered in dedicated on-chip memories and register banks to enable efficient data reuse. A top-level controller decodes instructions, generates memory addresses, and orchestrates data movement between buffers and the compute core. The compute core is organised as a systolic array of Processing Elements (PEs), where data flows through the array to perform convolution, pooling and activation operations in parallel across multiple lanes. The systolic input setup ensures correct temporal alignment of data to support continuous, high-throughput operation.

The partial and final results produced by the PE array are collected, reordered , and written into output buffers before being streamed back to DDR memory via the DMA engine. By combining streaming dataflow, on-chip buffering, and systolic execution, the design efficiently accelerates core convolutional neural network operations on the FPGA-based SoC.

1.3 Processing Element(PE) Architecture

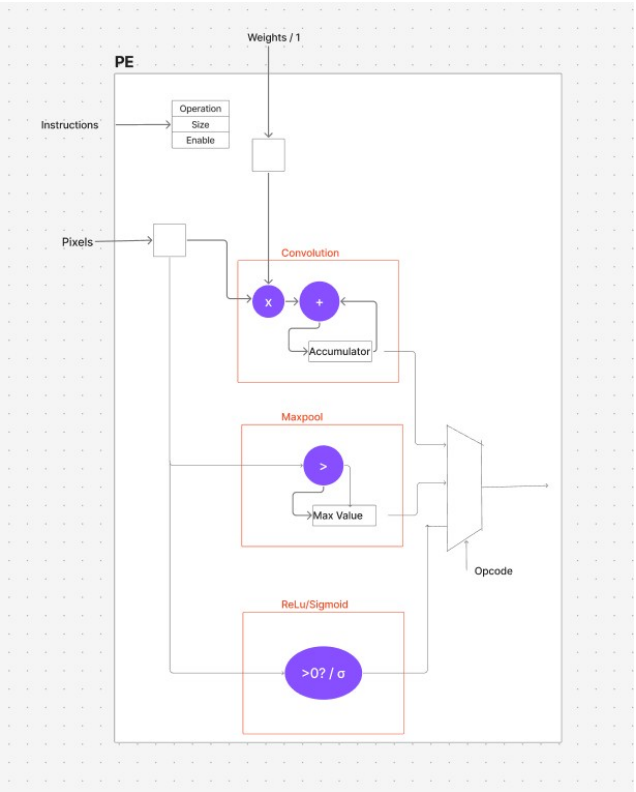

Figure 2: Processing Element (PE) architecture of the SIMD processor array

The Processing Element (PE) is the fundamental computational unit of the SIMD processor array and is designed to support convolution and pooling operations required for convolutional neural networks. Each PE operates on fixed-point data and includes a pipelined multiply–accumulate (MAC) datapath for convolution, as well as a comparison-based datapath for max-pooling. The PE accepts pixel values and corresponding weights as inputs and propagates these values to adjacent PEs, enabling efficient systolic-style dataflow. A mode control signal selects between convolution, max-pooling and activation functionality. The pipelined design ensures high throughput by allowing one operation to be initiated every clock cycle, making the PE well suited for parallel execution within the SIMD array.

1.4 Systolic Array Architecture

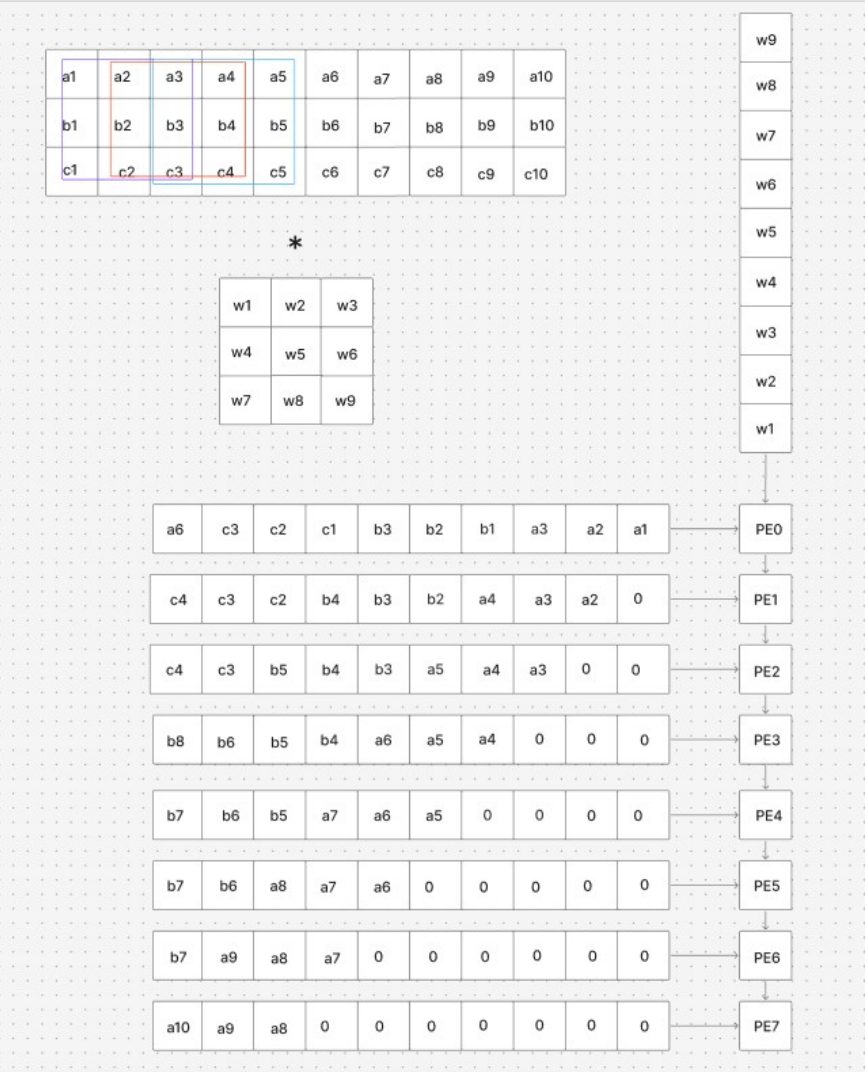

Figure 3: Systolic dataflow across an 8-PE processing element array

The systolic array comprises eight Processing Elements (PE0–PE7) arranged to enable a regular, pipelined dataflow for convolution and max-pooling. In this design, the input activation vector is provided as a word (multiple lanes), and a dedicated skewing stage (implemented using shift-register delays) time-aligns the lanes such that successive elements are presented with increasing cycle offsets. During operation, weight enters at the top PE and is propagated down the chain, while control signals such as clear are also forwarded PE-to-PE to synchronise accumulation across the column. Each PE produces a partial result per lane, and the array assembles the eight PE outputs into a single wide output word.

2 Implementation Results

3 Hardware Synthesis and Resource Utilization Report

The `simd_top` design was synthesized using Vivado 2024.2, targeting the Xilinx Zynq-7000 SoC platform. The following tables summarize the implementation details and hardware resource consumption.

3.1 Project Specifications

- **Target Device:** xc7z010clg400-1 (Zybo Z7-10)

3.2 Logic Resource Utilization

3.2.1 Systolic Input Control Module

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	110	0	0	17600	0.63
LUT as Logic	78	0	0	17600	0.44
LUT as Memory	32	0	0	6000	0.53
LUT as Distributed RAM	0	0			
LUT as Shift Register	32	0			
Slice Registers	196	0	0	35200	0.56
Register as Flip Flop	196	0	0	35200	0.56
Register as Latch	0	0	0	35200	0.00
F7 Muxes	0	0	0	8800	0.00
F8 Muxes	0	0	0	4400	0.00

Figure 4: Slice Logic

3.2.2 Register File

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	1	0	0	17600	<0.01
LUT as Logic	1	0	0	17600	<0.01
LUT as Memory	0	0	0	6000	0.00
Slice Registers	0	0	0	35200	0.00
Register as Flip Flop	0	0	0	35200	0.00
Register as Latch	0	0	0	35200	0.00
F7 Muxes	0	0	0	8800	0.00
F8 Muxes	0	0	0	4400	0.00

Figure 5: Slice Logic

2. Memory

Site Type	Used	Fixed	Prohibited	Available	Util%
Block RAM Tile	2	0	0	60	3.33
RAMB36/FIFO*	0	0	0	60	0.00
RAMB18	4	0	0	120	3.33
RAMB18E1 only	4				

Figure 6: Memory

3.2.3 Address Generator

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	168	0	0	17600	0.95
LUT as Logic	168	0	0	17600	0.95
LUT as Memory	0	0	0	6000	0.00
Slice Registers	90	0	0	35200	0.26
Register as Flip Flop	90	0	0	35200	0.26
Register as Latch	0	0	0	35200	0.00
F7 Muxes	0	0	0	8800	0.00
F8 Muxes	0	0	0	4400	0.00

Figure 7: Slice Logic

3. DSP

Site Type	Used	Fixed	Prohibited	Available	Util%
DSPs	1	0	0	80	1.25
DSP48E1 only	1				

Figure 8: DSP

3.2.4 Address Decoder

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	10	0	0	17600	0.06
LUT as Logic	10	0	0	17600	0.06
LUT as Memory	0	0	0	6000	0.00
Slice Registers	11	0	0	35200	0.03
Register as Flip Flop	11	0	0	35200	0.03
Register as Latch	0	0	0	35200	0.00
F7 Muxes	0	0	0	8800	0.00
F8 Muxes	0	0	0	4400	0.00

Figure 9: Slice Logic

3.2.5 Processing Elemnent

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	44	0	0	17600	0.25
LUT as Logic	44	0	0	17600	0.25
LUT as Memory	0	0	0	6000	0.00
Slice Registers	38	0	0	35200	0.11
Register as Flip Flop	38	0	0	35200	0.11
Register as Latch	0	0	0	35200	0.00
F7 Muxes	0	0	0	8800	0.00
F8 Muxes	0	0	0	4400	0.00

Figure 10: Slice Logic

3. DSP

Site Type	Used	Fixed	Prohibited	Available	Util%
DSPs	1	0	0	80	1.25
DSP48E1 only	1				

Figure 11: DSP

4 Simulation and Verification Results

The proposed accelerator supports convolution operations with kernel sizes ranging from 1×1 to 7×7, as well as max-pooling and ReLU activation. Timing diagrams and representative output results are shown below to demonstrate correct functionality and performance.

4.1 3X3 Maxpool

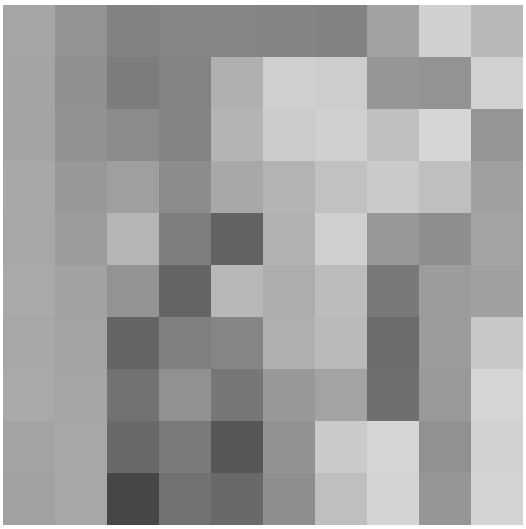

Figure 12: Python code output

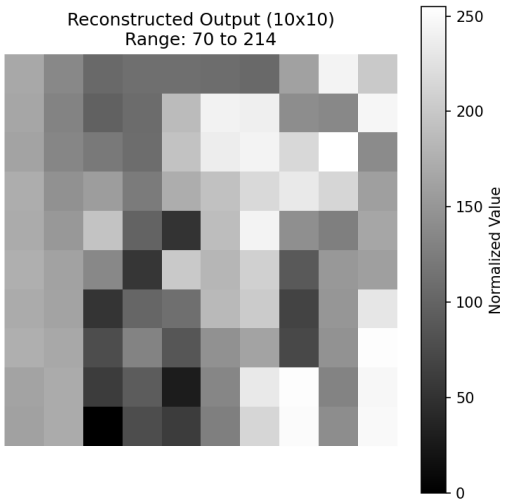

Figure 13: Vivado RTL output

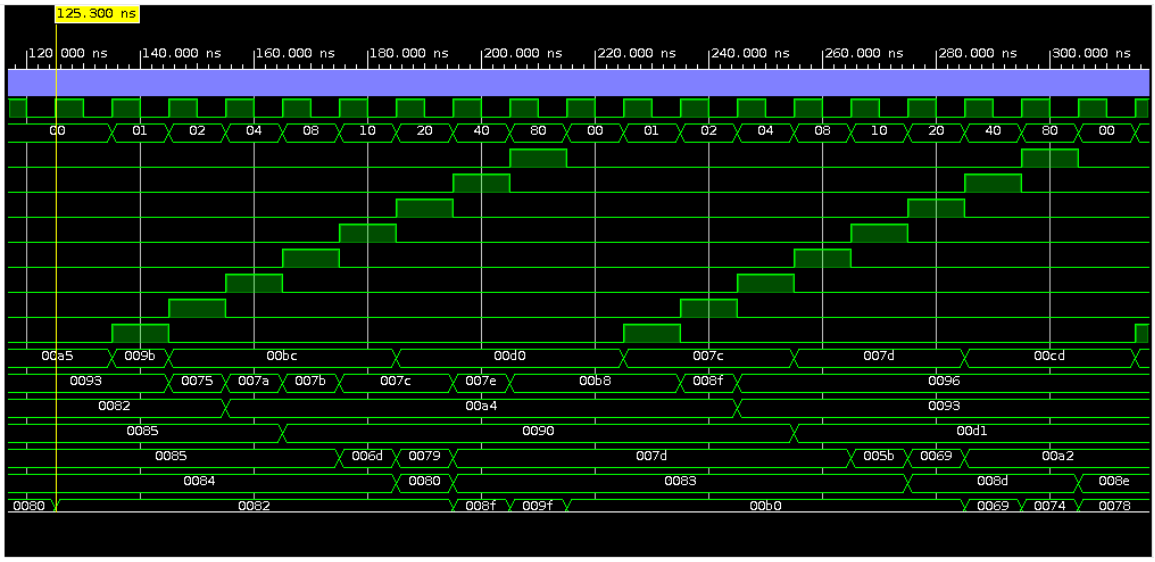

Figure 14: Timing Diagram for 3x3 max pool operation

4.2 2X2 Maxpool

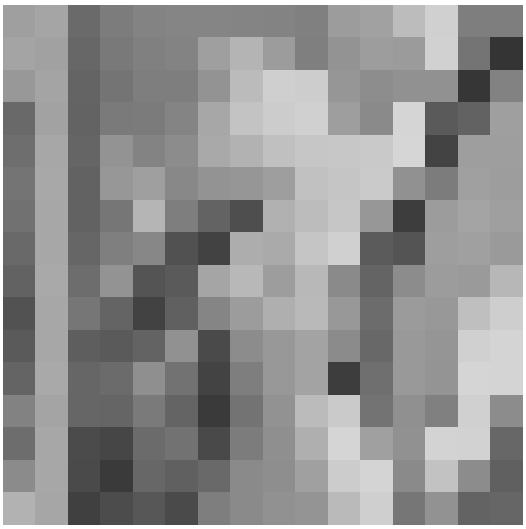

Figure 15: Python code output

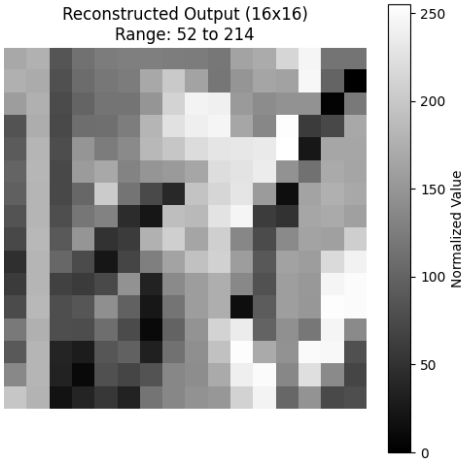

Figure 16: Vivado RTL output

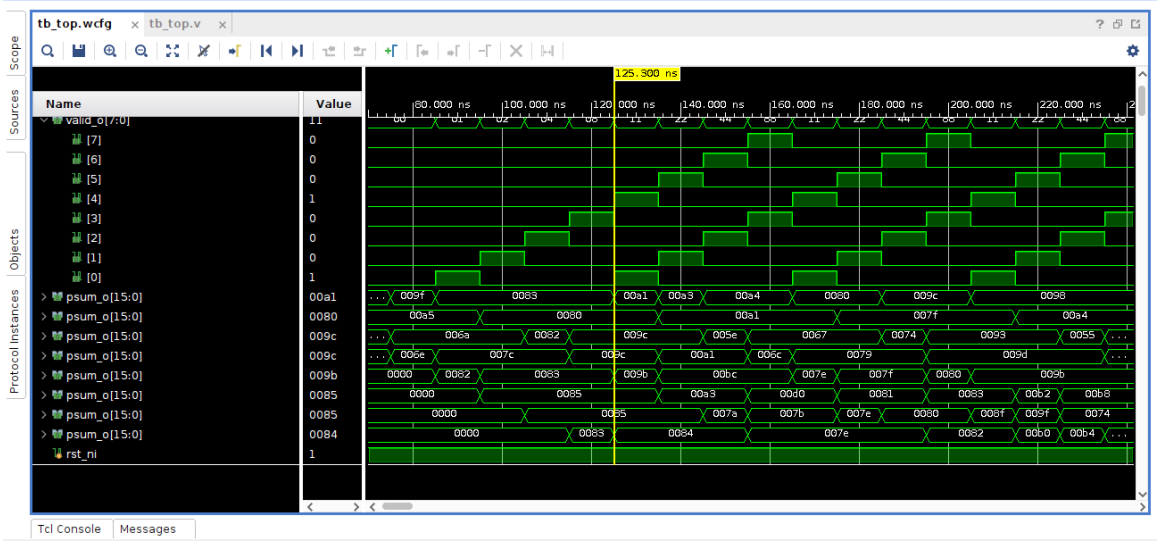

Figure 17: Timing Diagram for 2x2 maxpool operation

4.3 2X2 Convolution

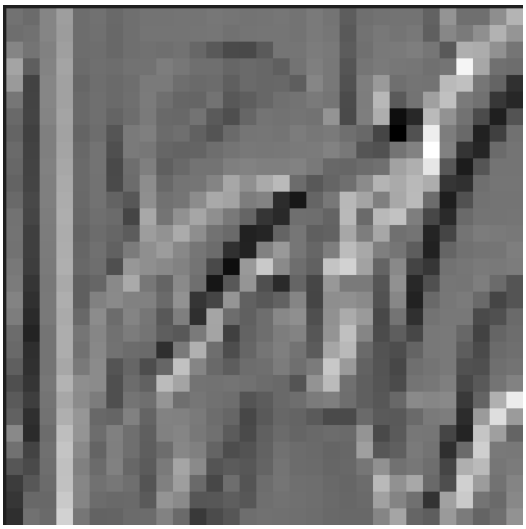

Figure 18: Python code output

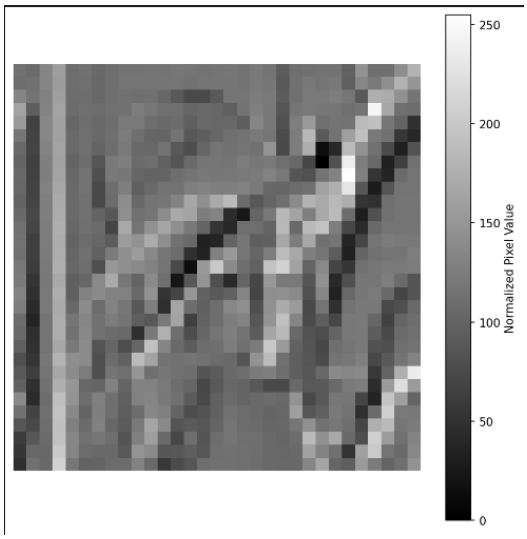

Figure 19: Vivado RTL output

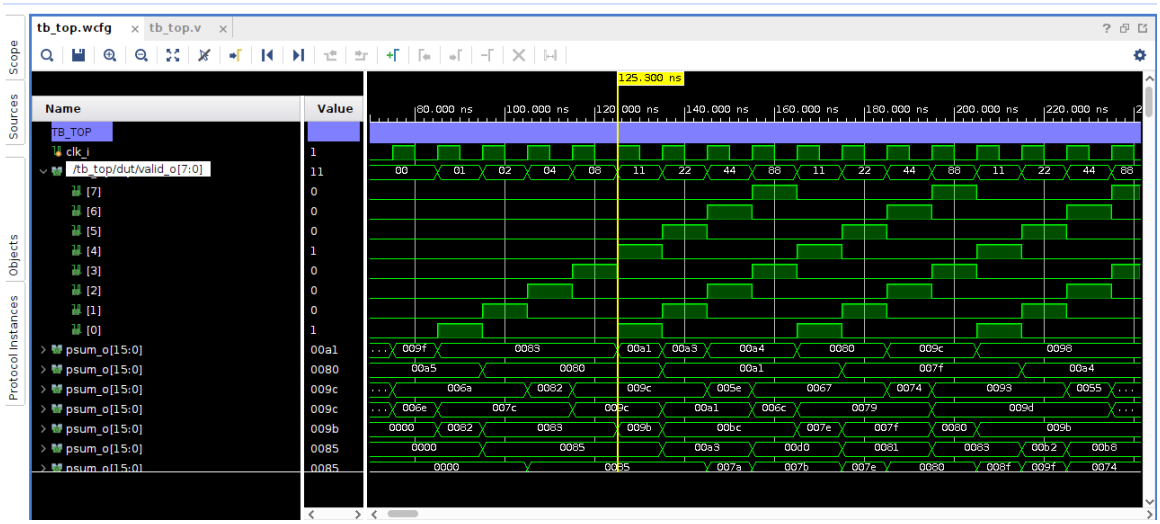

Figure 20: Timing diagram of 2x2 convolution

4.4 7X7 Convolution

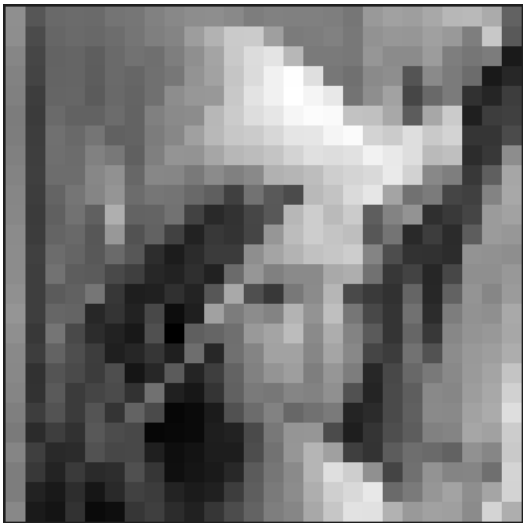

Figure 21: Python code output

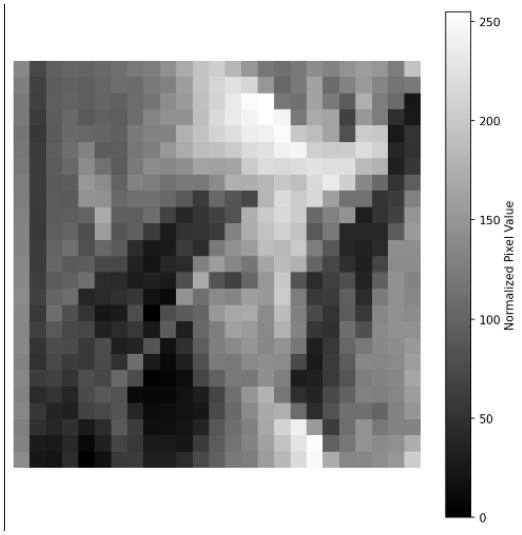

Figure 22: Vivado RTL output

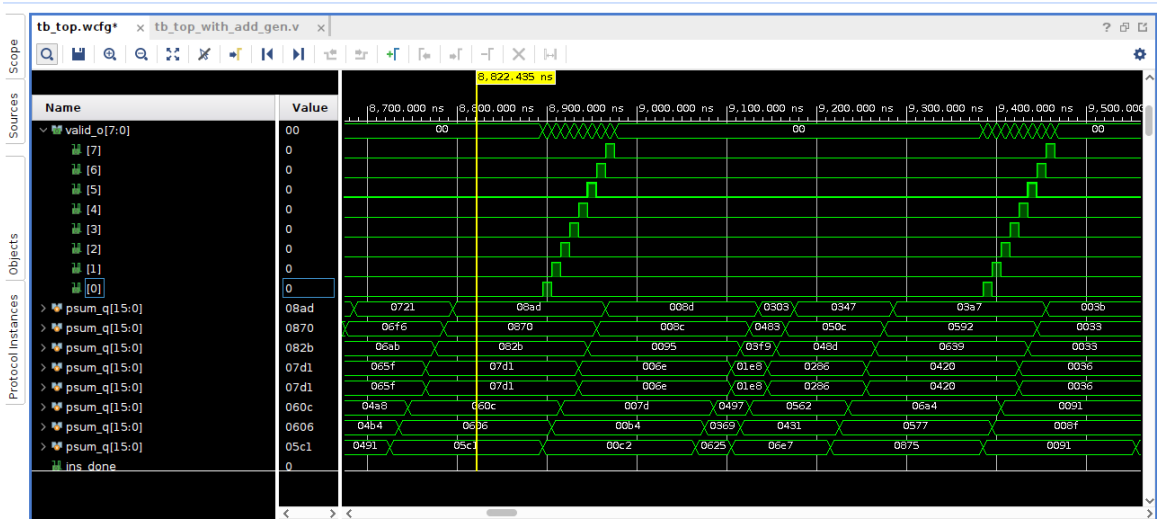

Figure 23: Timing diagram of 7x7 convolution

4.5 ReLu Operation

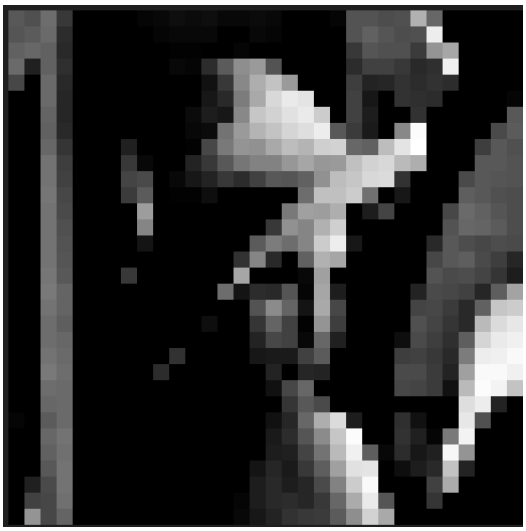

Figure 24: Python code output

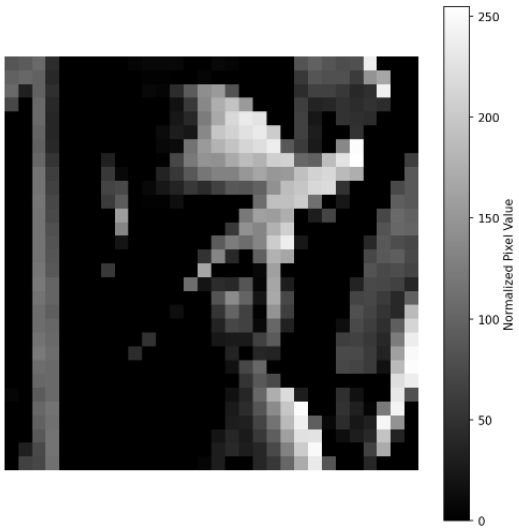

Figure 25: Vivado RTL output

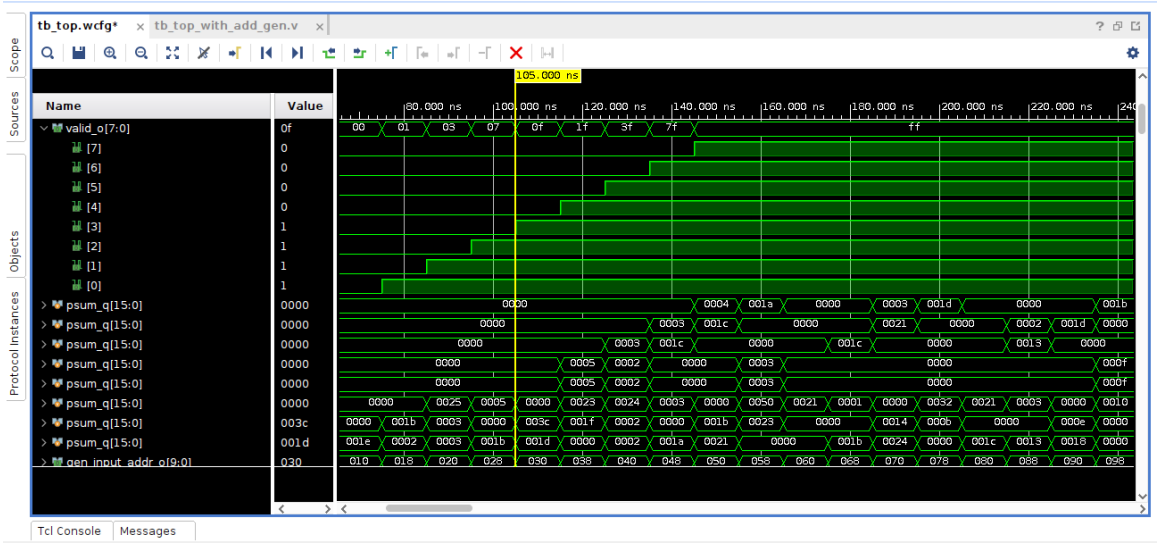

Figure 26: Timing Diagram for ReLu operation

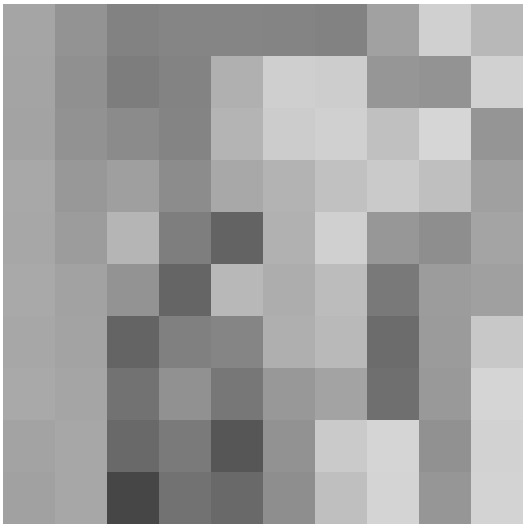

Figure 27: Python code output

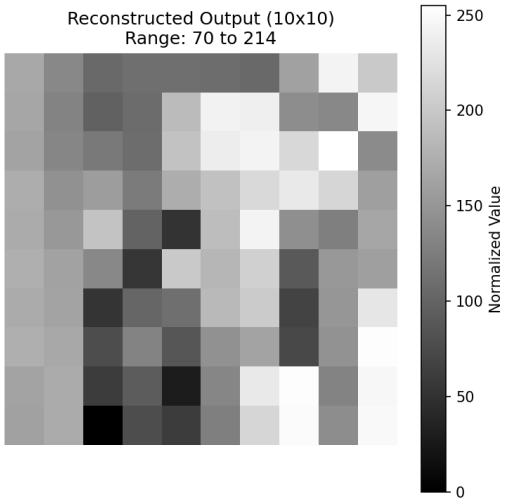

Figure 28: Vivado RTL output

4.6 Time taken for each operation

Table 3: Clock Cycles for Convolution Operations on 32×32 Image

Kernel Size	**Clock Cycles**
1×1	138
2×2	586
3×3	1090
4×4	1866
5×5	2810
6×6	3898
7×7	5106

Table 4: Clock Cycle Consumption for MaxPool Operations on 32×32 Input Image

Kernel Size	**Clock Cycles**
2×2	136
3×3	125
4×4	136

The ReLU activation function requires 138 clock cycles for processing the 32×32 feature map.

5 DMA-Based Image Processing Architecture

Figure 29 shows the proposed DMA-based image processing pipeline on the Zynq-7000 SoC, integrating UART image acquisition, software preprocessing, and high-throughput data transfer between the PS and PL using AXI DMA.

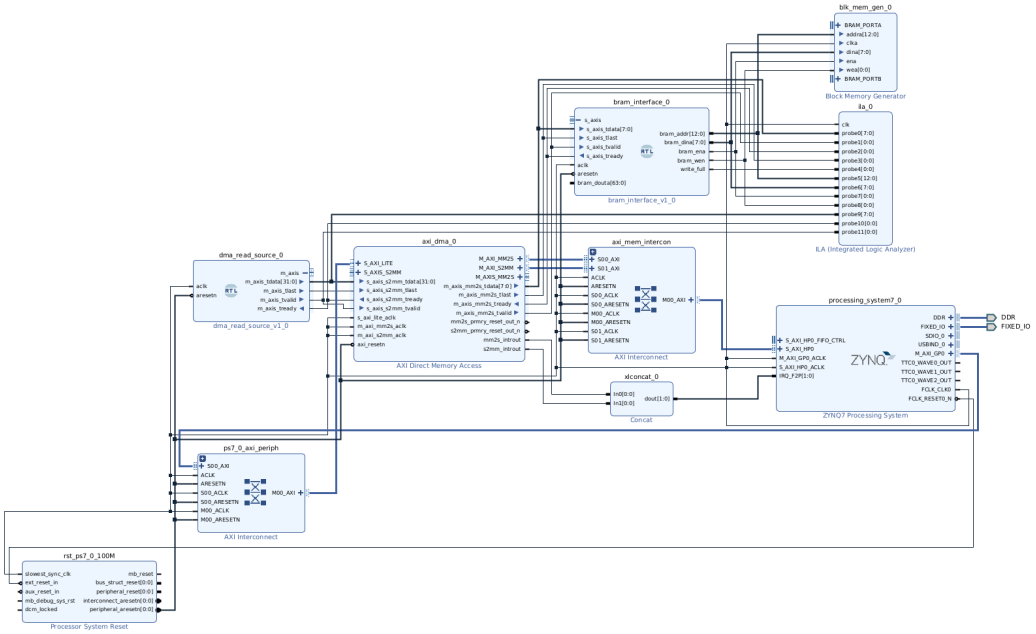

Figure 29: UART to AXI DMA based image processing architecture

5.1 System Overview

The PS executes an Embedded C application for UART image reception, preprocessing (*im2col*), and DMA control. The PL implements a streaming accelerator that processes AXI-Stream data and returns results to the PS via S2MM DMA.

The AXI DMA operates in *simple mode* with two channels:

- MM2S: DDR memory → PL
- S2MM: PL → DDR memory

Interrupts indicate transfer completion or errors.

5.2 UART-Based Image Acquisition

A 28×28 grayscale image is received byte-by-byte via UART into DDR memory, allowing flexible input without external storage.

5.3 AXI DMA Transmission (MM2S)

The preprocessed data is sent via MM2S DMA from DDR to the PL, freeing the CPU and enabling high-throughput transfer. Completion is signaled by an interrupt. Figure 30 shows an ILA capture of the continuous AXI-Stream transfer.

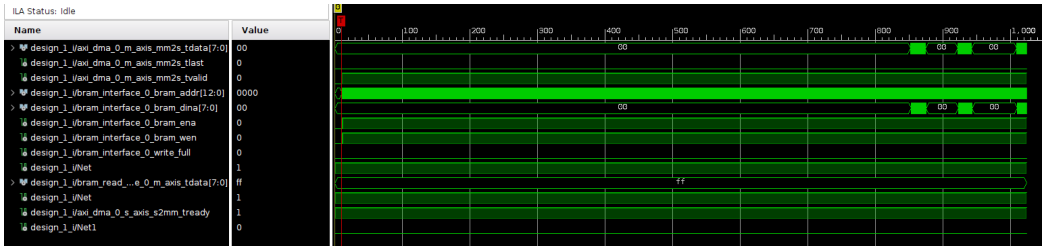

Figure 30: ILA capture of AXI DMA MM2S transfer

5.4 Hardware Processing and Reception (S2MM)

The PL processes the data and streams results back via S2MM DMA into a preallocated DDR buffer. An interrupt signals that valid output data is ready.

5.5 Interrupt Handling and Design Rationale

Separate ISRs handle transfer completion and errors for MM2S and S2MM channels, ensuring deterministic PS-PL synchronization. This architecture achieves:

- High-throughput AXI DMA transfers
- Minimal CPU overhead
- Modular PS-PL separation suitable for accelerator-based systems

It is well-suited for real-time image and signal processing on heterogeneous SoC platforms.