March 2020

Novel Bit-Sliced In-Memory Computing Based VLSI Architecture for Fast Sobel Edge Detection in IoT Edge Devices

Rajeev Joshi
University of South Florida

Novel Bit-Sliced In-Memory Computing Based VLSI Architecture for Fast Sobel Edge Detection in IoT Edge Devices

by

Rajeev Joshi

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Engineering
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Srinivas Katkoori, Ph.D.
Hao Zheng, Ph.D.
Mehran Mozaffari Kermani, Ph.D.

Date of Approval:
March 12, 2020

Keywords: Image Processing, Edge Computing, Binary Image

DEDICATION

In loving memory of my brother Manoj.

You are a champion.

We love you and we always will.

ACKNOWLEDGMENTS

I would like to express my sincere thanks to Dr. Srinivas Katkoori for giving me an opportunity to work on this fascinating project. I am eternally grateful to him for his constant guidance and support and for giving me valuable suggestions throughout my thesis work. I would also like to thank Dr. Hao Zheng and Dr. Mehran Mozaffari Kermani for being on my thesis committee.

My sincere gratitude towards God, my family, and my gurus for their invaluable support, unconditional love, and motivation in every phase of my life. I appreciate my friends Adnan, Love, Arif, Hernan, Rubel, Sami and Sapana for their support in this endeavor.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

For today's Internet-of-Things (IoT) edge devices, there is an acute need for fast and power-efficient hardware for an image processing task. Traditional hardware solutions with sequential and/or pipelined architectures incur high latency and power. This motivates us to propose a novel in-memory computing architecture for rapid image processing. We propose a bit-sliced in-memory computing architecture for CMOS VLSI implementation for fast Sobel edge detection. To the best of our knowledge, this is the first work to propose in-memory computing based VLSI architecture for edge detection. The novelty of the proposed work is that one image can be processed in constant time irrespective of the image size. Binary images are used as input to the design. The Sobel operator equations are simplified by operator strength reduction, bit manipulation, and common term sharing across equations. The captured image is loaded into the design and all block-level operations are executed in parallel close to where the data resides. The architecture is highly modular and can be scaled for any image size. The block processing element (PE) is implemented at the layout-level with the Synopsys tool suite. For processing one block frame (3 x 3 pixel block) in 90 nm CMOS technology node, the number of logic gates is 17 with a worst-case delay of 3.52 fs and a total bounding box layout area of 158 nm^2. The estimated average power dissipation is 0.72 μW at 0.7 V supply voltage.

CHAPTER 1: INTRODUCTION AND MOTIVATION

As with the advancement of internet and digital technology, the Internet of Things (IoT) is becoming an integral part of human day to day life. Statista [1] estimates the number of IoT connected devices will go as high as 75.44 billion worldwide by 2025 which will be a five-fold increment within a decade as shown in Figure 1.1. Cisco Internet Business Solutions Group (IBSG), 2011 [2], also prognosticated the number of IoT connected devices would be 25 billion by 2015 and 50 billion by 2020. Today the digital image is becoming an essential artifact from social sites to the major research. Digital image processing [3] is thus creating a profound impact in every technical field. The digital image processing is being interconnected in IoT edge nodes is a hot topic that has several applications in almost every field of work such as medicine, transportation, space, biology, etc.

IoT devices use different kinds of sensors that work together in a synergistic manner. Image sensors have become one of the intrinsic parts of those IoT connected devices. With the growth of data globally in no time, the image and video data are becoming the most prominent artifacts which need efficient processing technology. The reason why IoT connected devices are deployed in every field of work for image sensing is due to ease of use, ease of installment, pretty accurate results, high efficiency, and affordability. But, there are limitations and issues related to image processing with IoT connected devices such as fast and efficient real-time image and video processing architectural design which are power and energy-efficient with less memory consumption and less area overhead.

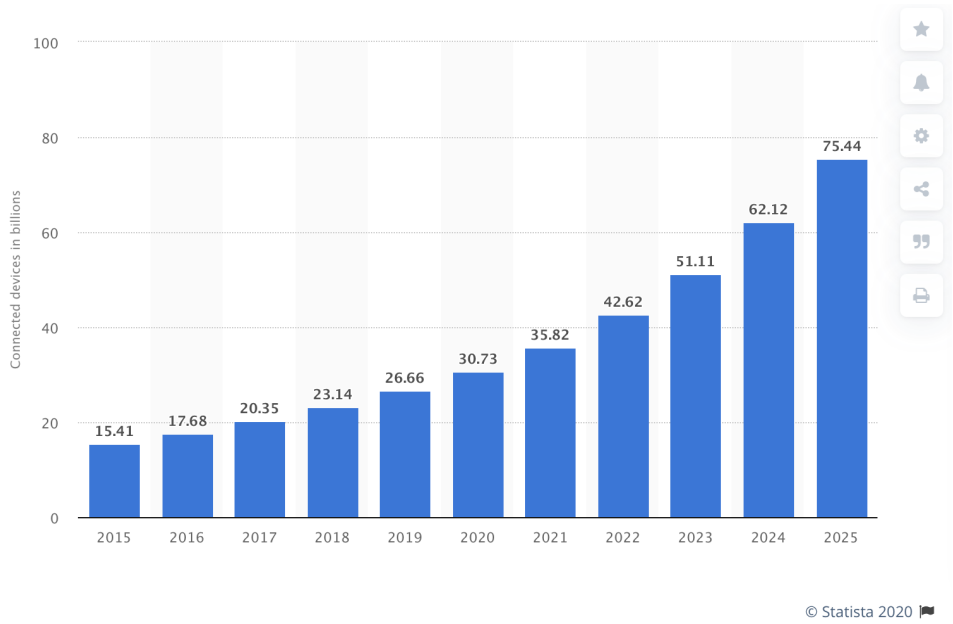

Figure 1.1: IoT Connected Devices Globally 2015-2025. (Reproduced from [1])

1.1 Proposed Approach

We propose a fast in-memory CMOS VLSI bit-sliced 2D architecture for the Sobel edge detection technique for digital image processing. The proposed architecture uses a binary image as the input as they require less memory for storage. Bit manipulation technique is used to simplify the computation process at the gate level for the image using Sobel operators in x-direction and y-direction. Each pixel is represented as a single bit either 0/1. For operator implemented in either x- direction or y-direction, there are total 8 one-bit inputs which are used for the computation of the final edge detected pixel. Once the single-bit output is computed for both the directions, the resultant single bit output is calculated by ORing the results from both directions. The final single bit output is stored at the center of the neighborhood pixel on which the Sobel filters are operated. Once the computation is performed on all the pixels, the architecture produces the edge detected image. As the architecture involves bit manipulation at the gate level, there is an on-site

computation and storage of the bits in a layered fashion which makes this architecture energy and power efficient with high performance and less area overhead suitable for implementation of the proposed design.

1.2 Experimental Results

We validated our proposed architecture on several binary images such as MNIST handwritten digits from 0-9, English alphabet, and different shapes of objects. We compared our results with the software implementation of Sobel edge detection on binary images with MATLAB software. We found that the edge detected images produced by our proposed architecture is consistent and accurate with a low error percentage in pixels as compared with the MATLAB implemented edge detected image. Also, the proposed architecture shows high performance, high energy and power efficiency as compared to the state-of-the-art CMOS VLSI based implementation of the Sobel edge detection.

1.3 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 presents a brief survey of the state-of-the-art digital image processing, digital image processing techniques, Internet of Things (IoT) and image processing, CMOS VLSI based edge detection architecture, and In-memory computing. Chapter 3 presents in detail the proposed fast Sobel edge detection architecture and realization of the Sobel edge equations. Chapter 4 reports the experimental setup, experimental results, and the analysis and discussion. Chapter 5 draws the conclusion and outlines the direction towards future work for further enhancement.

1.4 Summary

In this Chapter, we discussed the necessity of the optimized architecture for the digital image processing for IoT connected devices. We also motivate the need for considering the in-memory CMOS VLSI bit-sliced architecture for digital image processing. The scope and overall overview of the thesis work are presented.

CHAPTER 2: RELATED WORK

Various research works have been done related to the implementation of different image processing techniques in hardware. In this Chapter, we review the contemporary works related to image processing and different edge detection techniques, CMOS VLSI and bit-sliced based architectures, IoT and In-memory computing in detail.

2.1 Digital Image Processing

An image can be defined as a function *I(a,b)* in a two-dimensional plane, where a and b are the co-ordinates [3]. An image has an amplitude which is also called intensity or gray level at a particular point. A digital image is an output of electronic devices such as cameras or scanning devices. It has finite number of basic elements with a specific location and values which are called pixels. A digital image is sampled and mapped using these pixels.

The term digital image processing dealt with the processing and manipulation of digital images to enhance or extract useful information via digital computers. Each pixel in an image has a tonal value which is coded using binary values i.e., 0 and 1. Bit depth defines the numbers of tones in an image. Based on bit depth, there are three kinds of digital images.

- Black and White (bi-tonal) image: In this type of image, each pixel is 1-bit in size and produces dual tones, black and white, 0 for black and 1 for white.
- Gray-scale image: Each pixel consists of multiple bits to represent different tones of an image.

Usually, the gray-scale is represented using 2 to 8 bits per pixel.

- Color image: A color image is represented by more than 8 bits per pixel to show the image information. An increase in bits per pixel produces more color combinations in a color image.

There are many image processing techniques that are used for various purposes such as image filtering, image segmentation, image restoration, etc. In this work, our focus is on edge detection therefore, we will discuss the edge detection techniques.

2.2 Edge Detection Techniques

In an image, the edge can be defined as a sharp change (or discontinuity) in image intensity which occurs at the boundary between two separate regions [4]. There are different types of edges in an image. The step edge is the sudden change in intensity value from one point to another. The line edge is a sudden change in intensity value which restores back to the initial intensity value after a certain distance. The ramp edge refers to the constant change in image intensity value. Roof edge refers to an increase and decrease in intensity over a period of time.

Edge detection is one of the basic steps in digital image processing in computer vision. Edge detection is a method of detecting the regions or boundaries in an image where there are sharp changes in intensity/color in which a higher intensity value indicates a steep change and lower intensity value indicates a shallow change in an image. The main purpose of edge detection in an image is to reduce the amount of data in an image and to filter out the unnecessary information from an image. Edge detection helps to gather significant information from any image with less memory required for processing and storage. There are various types of edge detection techniques for digital image processing. The edges in an image can be detected by computing the discrete

gradient approximation.

The gradient or first derivative can be defined as a vector as follows:

$$G = \begin{pmatrix} G_x \\ \\ G_y \end{pmatrix} = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \\ \frac{\partial f}{\partial y} \end{pmatrix} \tag{2.1}$$

The magnitude and direction of the gradient are given by:

$$|G| = \sqrt{G_x^2 + G_y^2} \tag{2.2}$$

$$\Theta = \arctan(G_y/G_x) \tag{2.3}$$

The gradients in Equation 2.1 can be approximated as follows:

$$G_x \approx f(i, j+1) - f(i, j) \tag{2.4}$$

$$G_y \approx f(i, j) - f(i+1, j) \tag{2.5}$$

In this section, we will discuss some of the widely used gradient edge detection techniques.

2.2.1 Roberts Edge Detection

The Roberts Cross operator is one of the first edge detectors. This is a simple and quick differential operator that computes the two-dimensional spatial gradient of an image by applying discrete differentiation [5].

This operator uses two 2 x 2 kernels or filters, one in the x-direction and another in the

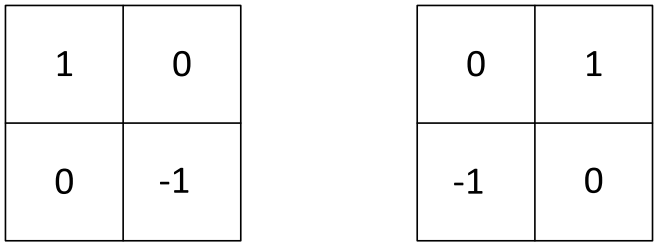

Figure 2.1: 2 x 2 Kernels of Roberts Operator

y-direction to compute the edges in an image. The kernels are shown in Figure 2.1. As we can see, the kernels are 90° rotation of each other. Roberts Cross operator performs convolution over the image using these two kernels to detect the edges in an image.

Let us say *I(x,y)* be any arbitrary point in an image. When Roberts Cross operator is convolved over the x-direction and y-direction, we will get the gradient $G_x(x, y)$ and $G_y(x, y)$in the respective directions.

The magnitude of the gradient is given by:

$$|G| = \sqrt{G_x^2 + G_y^2} \tag{2.6}$$

and the approximate gradient is computed as:

$$|G| = |G_x| + |G_y| \tag{2.7}$$

The direction of the gradient is given by:

$$\Theta = \arctan(\frac{G_y}{G_x}) - \frac{3\pi}{4} \tag{2.8}$$

However, this operator does not give information about the edge orientation and is usually affected by noise [6].

2.2.2 Prewitt Edge Detection

Prewitt operator is a discrete differential operator that compute the edges in an image by taking the difference between the corresponding pixel intensities in the image [7]. This operator detects edges in the horizontal and vertical direction using two types of 3 x 3 kernels. Figure 2.2 shows the Prewitt kernels. The Prewitt operators are simple and give information about the image orientation. These kernels are convolved over the image to obtain the edges in an image using differentiation of the pixel intensities.

-1	0	1
-1	0	1
-1	0	1

-1	-1	-1
0	0	0
1	1	1

Figure 2.2: 3 x 3 Kernels of Prewitt Operator

Let us say *I(x,y)* be any arbitrary point in an image. When the Prewitt operator is convolved

over the x-direction and y-direction, we will get the gradient $G_x(x, y)$ and $G_y(x, y)$in the respective directions.

Mathematically, the magnitude of the gradient is calculated as:

$$|G| = \sqrt{G_x^2 + G_y^2} \tag{2.9}$$

and the approximate gradient is computed as:

$$|G| = |G_x| + |G_y| \tag{2.10}$$

The direction of the gradient is given by:

$$\Theta = \arctan 2(G_y/G_x) \tag{2.11}$$

2.2.3 Sobel Edge Detection

One of the common edge detection technique is Sobel edge detection. This technique computes the first-order derivative of an image and hence calculates the difference of the pixel intensities at the edges [8]. The technique uses two operators or kernels of 3 x 3 matrices, one in the x-direction and another in the y-direction. These kernels are convolved over the original image to get the edge-detected image. Figure 2.3 shows two Sobel operators in the x-direction and y-direction.

These kernels are convolved separately over an image to compute the gradient in each direction and then combine together to find the absolute magnitude of the gradient at that point of the image.

-1	0	1
-2	0	2
-1	0	1

-1	-2	-1
0	0	0
1	2	1

Figure 2.3: 3 x 3 Kernels of Sobel Operator

Let us say *I(x,y)* be any arbitrary point in an image. When the Sobel operator is convolved over the x-direction and y-direction, we will get the gradient $G_x(x, y)$ and $G_y(x, y)$ in the respective directions.

Mathematically, the magnitude of the gradient is calculated as:

$$|G| = \sqrt{G_x^2 + G_y^2} \tag{2.12}$$

and the approximate gradient is computed as:

$$|G| = |G_x| + |G_y| \tag{2.13}$$

The direction of the gradient is given by:

$$\Theta = \arctan(G_y/G_x) \tag{2.14}$$

The working of Sobel edge detection algorithm is explained as follows: The image is taken and we convolve horizontal and vertical kernel of Sobel operator to the original image to calculate the approximate gradient $G_x(x, y)$ and $G_y(x, y)$ for each pixel in both horizontal and vertical direction respectively. Then, combining both the approximate gradients in x and y direction, we compute the overall magnitude and direction of the gradient at each pixel in an image. This results in the edge detected pixel value. To compute the approximate gradient for all the pixels in an image, the Sobel kernels are moved over all the pixels in an image, computing one pixel at a time and then shifting the kernel window to the right by one pixel. Once the end of the row is reached, the kernels are moved to the beginning of the next row. We iterate this process for all the pixels in an image to compute the overall edge detected image.

In this work, we propose 2D hardware architecture for implementing the Sobel edge detection technique. We will present in detail the architecture and implementation in Chapter 3.

2.3 CMOS VLSI Based Architectures for Sobel Edge Detection

Various CMOS VLSI based architectures are proposed by researchers for Sobel edge detection. In this section, we will review some of the major works done in the literature. Kanopoulos, Vasanthavada, and Baker [9] proposed a design and implementation of the Sobel edge detection in CMOS technology on a single chip. The proposed architecture is validated in 2 μm CMOS technology node and implemented on silicon compiler system. The design is highly pipelined which can perform 200 x 10^6 per second for computing the output image gradient and direction. The design is operated at a frequency of 10 MHz with two-phase clock.

Boo, Antelo, and Bruguera [10] proposed a CMOS based design and implementation of the

edge detection of images using Sobel operators in an Application Specific Integrated Circuit (ASIC) using systolic processors arrays for the efficient architectural design. The proposed design is simple and modular which uses carry-save adder arithmetic to improve the performance of the architecture. The architecture was implemented in 1 μm CMOS technology node with a total area of 10 mm^2. The design works at a frequency of 50 MHz and produces a row and column-wise pixels of the gradient image alternatively in each clock cycle with a latency of 20 clock cycles. This design is validated on gray-scale images of size 512 x 512. The number of transistors and gates used is 27,340 and 6,835 respectively.

Kazakova, Margala, and Durdle [11] proposed CMOS based efficient design and implementation of a Sobel edge detection processor which was created as a part of the volume rendering system for computing the gradients and directions for different image applications. The design of the Sobel edge detection processor was implemented in 0.18 μm CMOS technology node. Wallace compression tree and carry-select adders are used to design the Sobel processor for computing the gradients. Pipelining and parallelism are implemented at the component level for improving power efficiency and performance. The design employed reduced swing complementary pass transistor logic which improves the performance further. The paper claims that the simulation results show a worst-case delay of 4.61 ns with an average power dissipation of 8.24 mW when operated with 200 MHz at 1.8V supply voltage.

2.4 IoT and Image Processing

Internet of Things (IoT) refers to number of physical devices which are connected together via internet network which can collect, process and share data among themselves in real-time without

the human intervention. The recent advancement in technology in regards to low cost powerful tiny hardware chips and ubiquitous wireless networks has made this possible. The research and advancements in IoT technology are making the world around us smarter and intelligent connecting the physical world with the digital environment.

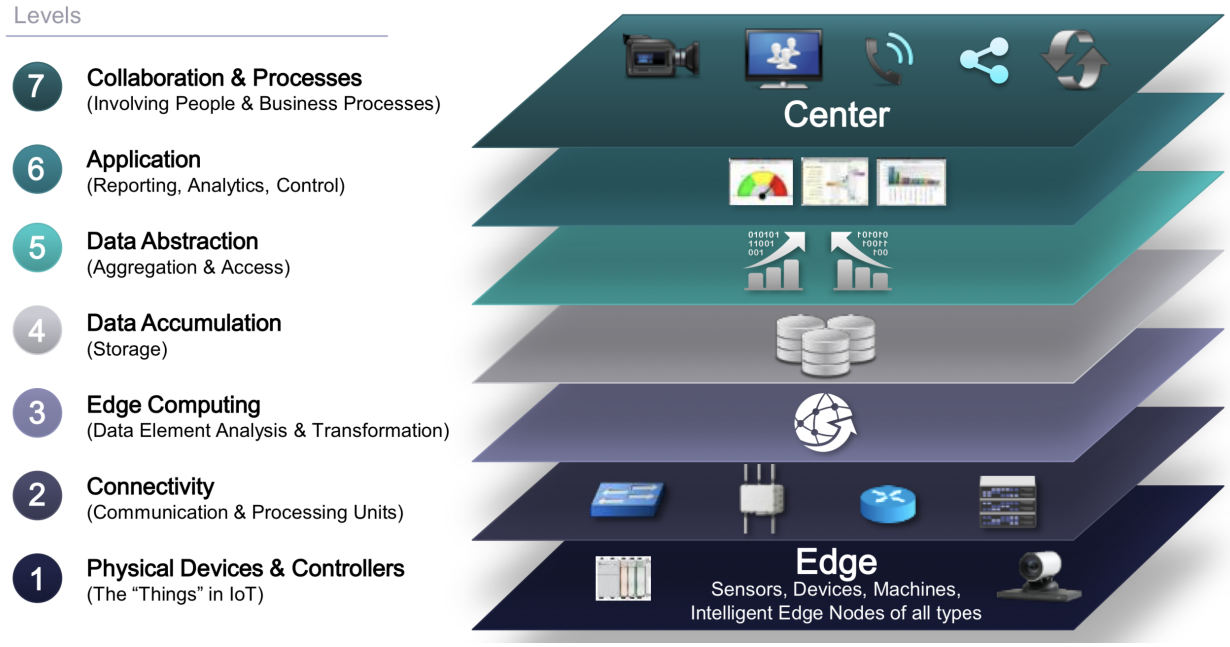

Figure 2.4: IoT Reference Model Published by the IoTWF Architectural Committee. (Reproduced from [11])

Figure 2.4 shows a seven-layer IoT reference model published by the IoT World Forum (IoTWF) architectural committee in 2014 [12]. The different layers of the IoT reference model are described below.

- Physical Devices and Controller Layer: The layer includes different kinds of endpoint devices and sensors that can exchange the information.
- Connectivity Layer: The main function of this layer is to make reliable communication across the network.

- Edge Computing Layer: The edge computing layer performs the reduction and reformatting of the data which is processed by layers at a higher level.
- Data Accumulation Layer: The data accumulation layer performs the conversion of event-based data to query-based data for storage.
- Data Abstraction Layer: The data abstraction layer checks the compatibility of different data formats coming from different sources and aggregates the data into one place.
- Application Layer: The application layer consists of software applications to analyze the data.
- Collaboration and Processing Layer: The collaboration and processing layer involves the collaboration and sharing of application information.

In recent years, lot of research work has been done to synergistically combine IoT and image processing tasks and applications. Dorothy, Kumar, and Sharmila [13] present an overview of IoT based automatic systems for home security using digital image processing algorithms. The method uses sensors and cameras together with edge devices. The proposed work uses template-based methods such as grey-scale based matching and edge-based matching and Fast Fourier Transform (FFT) along with twiddle factors for receiving and comparing the images with the database stored in nodes. These image processing algorithms are used to process the captured image in order to validate it with the stored database in nodes. Tseng et al. [14] integrated IoT with image recognition system for developing the efficient home-delivered meal services for the elderly people by combining the statistical histogram and k-means clustering for image segmentation. Frank et al. [15] proposed an IoT based smart traffic signaling system that measures the traffic density using the video or

image processing. In order to determine the traffic density, the captured images are compared to the database stored at the server.

Kapoor et al. [16] integrated IoT and image processing techniques together to develop a smart agricultural system using various sensors, cameras along with Aurdino board for monitoring and to observe the leaf lattice, environmental factors and other human interventions such as the use of fertilizers and pesticides for proper growth of the plants. Rane, Dubey, and Parida [17] proposed an IoT based vehicle parking system using microcontroller, cameras, and image processing techniques to solve the traffic congestion problem for managing the parking spaces efficiently. The proposed work uses OCR (Optical Character Recognition) image processing algorithm to process the registered vehicle number and similarly, the parking space information are sent to the server through the microcontroller. User can use a mobile app to get the information about the available parking space any time through the server.

2.5 In-memory Computing

Today the most common architecture used for almost all the computing in the world is the Von Neumann architecture. Von Neumann architecture is based on the principle of movement of data between the processor and the memory. As in recent years, the processors speed is increasing dramatically as compared to the memory speed which is significantly affecting the throughput. Due to this mismatch in bandwidth between the fast CPU and slow memory, Von Neumann architecture suffers a memory wall problem [18].

To overcome the above problem, there should be a significant change in the processor's architecture. One of the ways of alleviating this problem is In-memory computing. The core idea

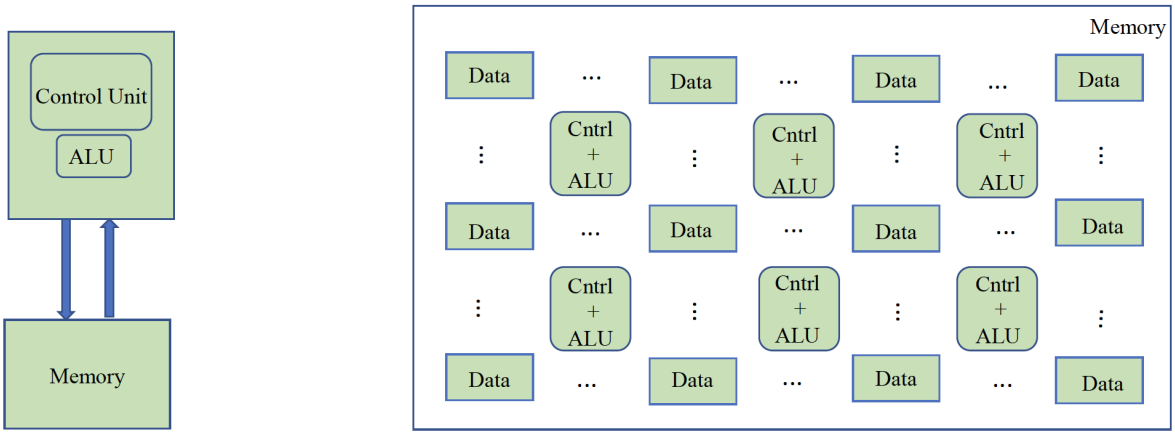

Figure 2.5: Data Processing Architecture (a) Von Neumann and (b) In-memory Computing

here is to bring processor and memory close to each other. In-memory computing can be defined as a computation in which tasks are processed near or inside the memory.

Figure 2.5 shows the Von-Neumann and in-memory computing architectures. The in-memory computing architecture consists of the memory elements and the processing elements (PE). The memory elements are used to store the data and the processing element (PE) perform computation on the data which are distributed inside the memory where data resides.

In this work, we proposed an in-memory CMOS VLSI based architecture for Sobel Edge Detection. This improves the throughput and overall performance of the architecture.

2.6 Summary

In this Chapter, we reviewed and summarized different works that have been done related to image processing and various edge detection techniques. We also reviewed hardware architectural design and implementation of the Sobel edge detection technique, and IoT and image processing.

CHAPTER 3: BIT-SLICED IN-MEMORY COMPUTING BASED VLSI ARCHITECTURE FOR FAST SOBEL EDGE DETECTION: PROPOSED METHOD

We present in detail the proposed bit-sliced in-memory computing VLSI based architecture for fast Sobel edge detection. The overall idea is as follows: Given colored or grayscale image, we convert it into a binary image. The raw version of a binary image is obtained which is used as an input to the design. The architecture processes the given image and produces an edge detected image in a raw format. This raw format is later converted to the standard binary image format.

This chapter is organized as follows: Since the input image needs to be convert into a raw file format, first we will discuss how input image is processed and converted to a raw image file. We will then describe the proposed CMOS VLSI based 2D architecture for Sobel edge detection. We will also explain how the Sobel operators' equations are optimized for in-memory computing. Finally, this idea is illustrated using a simple example.

3.1 Input Image Processing

In the proposed architecture, a binary image is taken as an input. The colored or grayscale image is first converted to a binary image using MATLAB software as shown in Figure 3.1 [19] . The raw binary image format is obtained and then fed as an input to the proposed design. The image sizes are compressed as compared to the original image size. Before processing the image

(a) Color image (b) Greyscale image (c) Binary image

Figure 3.1: Types of Images. (Adapted from [19])

using the Sobel edge detection technique, the size of the compressed raw binary image is padded with zeros in all the rows and columns of the outermost pixel in an image. This increases the size of rows and columns of an image by 2.

3.2 Proposed In-memory CMOS VLSI Bit-Sliced Architecture

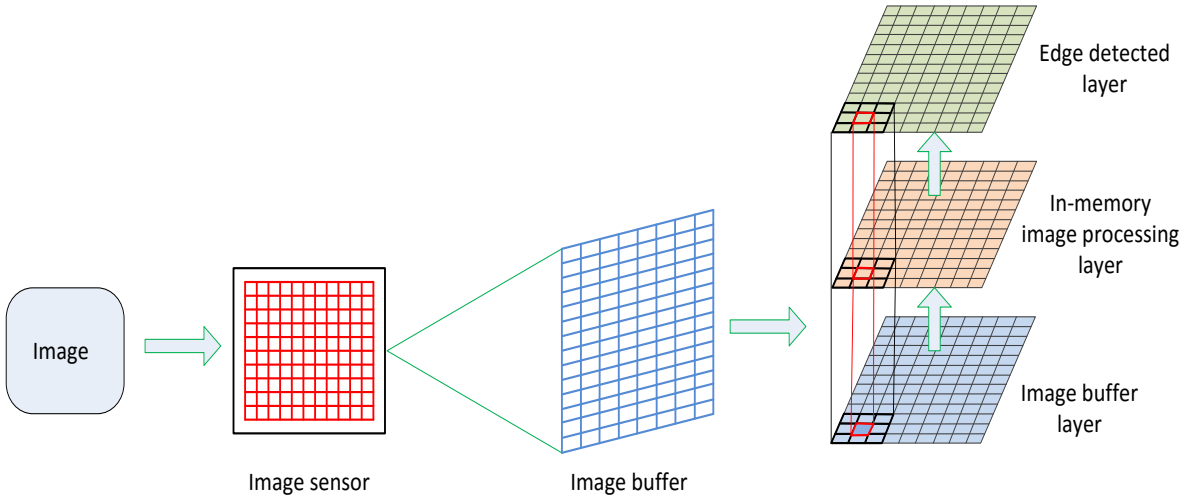

Figure 3.2: Three-Layered In-memory CMOS VLSI Architecture for Sobel Edge Detection

In this section, we will explain in detail the proposed in-memory CMOS VLSI bit-sliced

architecture for Sobel edge detection. Figure 3.2 shows the proposed model. The bit manipulation method is efficiently used to compute every edge detected pixel in the image. The two-dimensional bit-sliced model is presented. The image is detected and captured by the image sensor and generates a bitstream of raw image data. This bitstream of raw image data is fetched from the image sensor and stored in the image buffer array for processing and generating edge detected output image.

The processing of data is completed in a 3-layered structure as shown in Figure 3.2. The image buffer layer store the bitstream of raw image data before the computation and processing task. The in-memory image processing layer processes and computes the edge detected pixels of all the image data simultaneously. The architecture of the in-memory processing layer is shown in Figure 3.3.

The in-memory processing layer consists of memory elements and block processing elements (PE). The memory elements consist of D flipflops which are used to store all the raw image data which are processed by the block processing elements. The block processing elements consists of basic logic gates for computation and processing of the image data using operator strength reduction, bit manipulation, and common term sharing across equations. Each PE takes the eight input image data from the neighborhood memory elements and processes the bit stream efficiently to compute the edge detected pixel at that point and store the value back in the memory element. The computation in all the PEs is carried out simultaneously and the output gets stored in the edge detected layer.

The computation and storage of the pixel bit are performed *in situ* making the architecture very compact in design, efficient in performance, and power-efficient in computing the edges in an image. This makes the architecture very well suited for IoT based image applications.

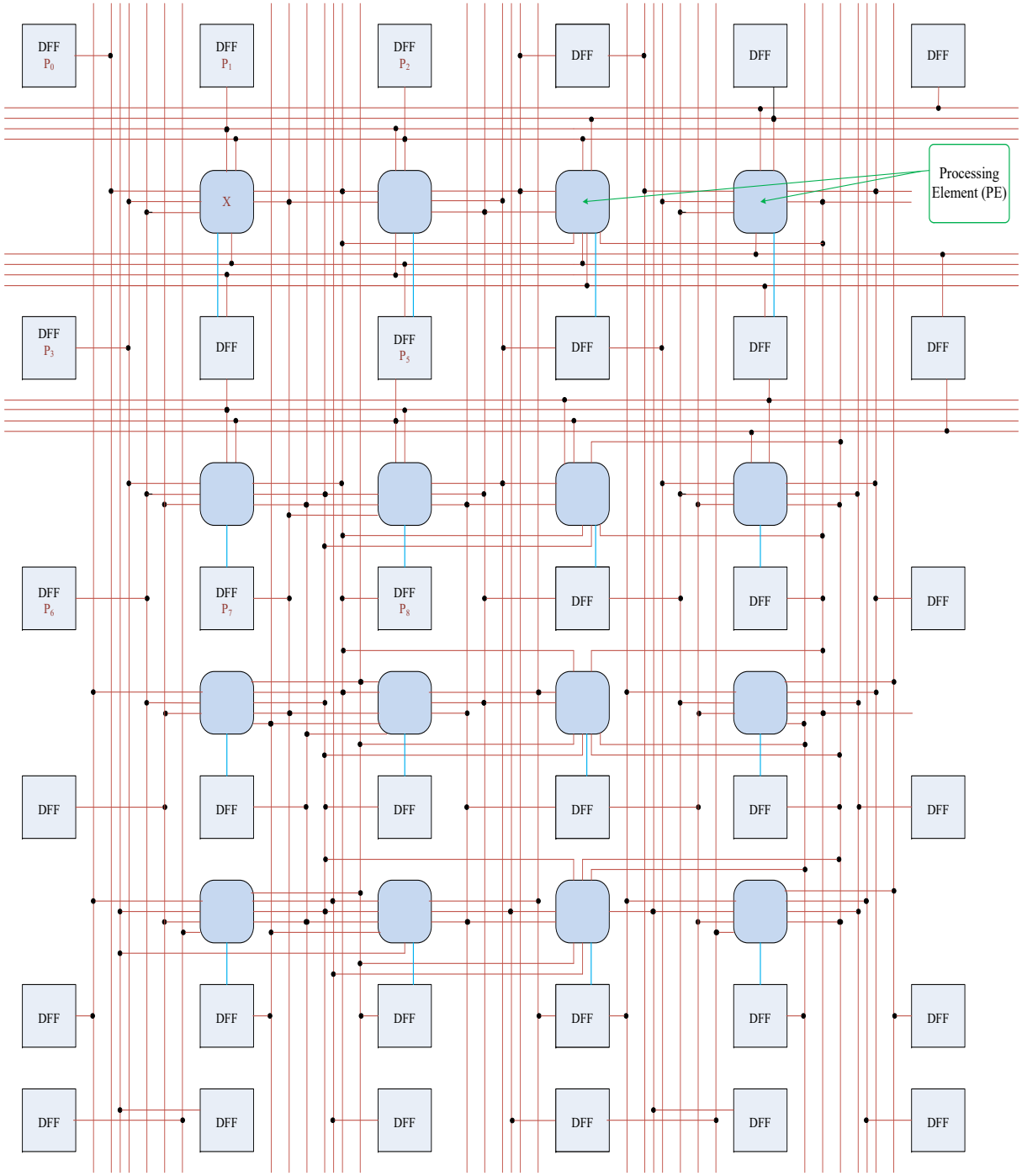

Figure 3.3: In-memory Processing Layer with D Flipflop Array and Sobel Edge Detection Block Array

3.2.1 Optimization of the Sobel Edge Operators for Efficient Hardware Implementation

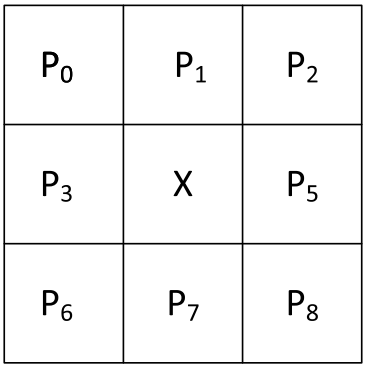

Figure 3.4: Pseudo-Convolution Kernel

To realize the Sobel edge operator in terms of hardware, we utilize the pseudo-convolution kernel for the simplification of the computation which is later realized using basic boolean logic gates. The pseudo-convolution kernel computes the approximate magnitude in both x-direction and y-direction of an image.

Figure 3.4 shows the pseudo-convolution kernel which is used for the computation of the middle pixel 'X' by performing the simple mathematical operations using the neighborhood pixels P_0, P_1, P_2, P_3, P_5, P_6, P_7, P_8.

The simplified arithmetic equations of the Sobel operator for computing a single edge detected pixel in both x-direction and y-direction are as follows:

$$G_x = (P_2 - P_0) + 2 * (P_5 - P_3) + (P_8 - P_6) \tag{3.1}$$

$$G_y = (P_6 - P_0) + 2 * (P_7 - P_1) + (P_8 - P_2) \tag{3.2}$$

The approximate magnitude of resultant G is given by:

$$|G| = |G_x| + |G_y| \tag{3.3}$$

The different mathematical operations in the pseudo-convolution kernel equations are simplified into different basic logic gates to take advantage of bit manipulation to compute the edge detected pixel bit in an image.

Let us consider Equation 3.1 and observe how we simplify this equation using different basic logic gates.

The multiplication by 2 on the second term in Equation 3.1 can be replaced by a single left shifting of the bit. Then, Equation 3.1 becomes,

$$G_x = (P_2 - P_0) + ((P_5 - P_3) << 1) + (P_8 - P_6) \tag{3.4}$$

Let us assume two bits are required to store the result of each term of the Equation 3.4. Here an extra bit is considered to store the result of a negative number in terms of 2's complement form. The terms can be simplified as follows:

$$t_1 t_2 = (P_2 - P_0) \tag{3.5}$$

$$t_3 t_4 = (P_5 - P_3) \tag{3.6}$$

$$t_5 t_6 = (P_8 - P_6) \tag{3.7}$$

We convert these terms into Boolean expressions using half subtractor logical expression which can be written as:

$$t_1 = (\overline{P_2} \wedge P_0) \tag{3.8}$$

$$t_2 = (P_2 \oplus P_0) \tag{3.9}$$

$$t_3 = (\overline{P_5} \wedge P_3) \tag{3.10}$$

$$t_4 = (P_5 \oplus P_3) \tag{3.11}$$

$$t_5 = (\overline{P_8} \wedge P_6) \tag{3.12}$$

$$t_6 = (P_8 \oplus P_6) \tag{3.13}$$

The left shifting of $(P_5 - P_3)$ by one bit is expressed as follows in Boolean expressions:

$$t_7 = (t_3 \wedge t_4) \tag{3.14}$$

$$t_8 = t_4 \tag{3.15}$$

$$t_9 = 0 \tag{3.16}$$

The addition of the terms $(P_2 - P_0)$ and $(P_8 - P_6)$ are simplified as follows:

$$x_1 = (t_5 \wedge t_6 \wedge \overline{(t_1 \oplus t_2)}) \vee (t_1 \wedge t_2 \wedge \overline{t_5} \wedge \overline{t_6}) \tag{3.17}$$

$$x_2 = x_1 \vee (\overline{t_1} \wedge t_2 \wedge \overline{t_5} \wedge t_6) \tag{3.18}$$

$$x_3 = (\overline{t_1} \wedge \overline{t_2} \wedge t_6) \vee (t_2 \wedge \overline{t_5} \wedge \overline{t_6}) \tag{3.19}$$

The addition of all the terms in the x-direction are simplified as follows:

$$G_{x_1} = (\overline{(x_1 \oplus x_2)} \wedge t_7 \wedge t_8) \vee (x_1 \wedge x_2 \wedge x_3 \wedge \overline{t_4} \wedge \overline{t_7}) \tag{3.20}$$

$$G_{x_2} = G_{x_1} \vee (\overline{x_1} \wedge x_2 \wedge x_3 \wedge \overline{t_7} \wedge t_8) \tag{3.21}$$

$$G_{x_3} = (\overline{x_1} \wedge x_2 \wedge t_8) \vee (x_1 \wedge x_2 \wedge x_3 \wedge \overline{t_4} \wedge \overline{t_7}) \tag{3.22}$$

$$G_{x_4} = (\overline{(x_1 \oplus x_2)} \wedge x_3) \vee \overline{t_7} \vee t_8 \tag{3.23}$$

The final term G_x is computed by logical ORing all the terms in x-direction, i.e.,

$$|G_x| = G_{x_1} \vee G_{x_2} \vee G_{x_3} \vee G_{x_4} \tag{3.24}$$

In a similar fashion, Equation 3.2 is simplified using different basic logic gates. The equations are shown as follows:

$$G_y = (P_6 - P_0) + ((P_7 - P_1) << 1) + (P_8 - P_2) \tag{3.25}$$

$$s_1 s_2 = (P_6 - P_0) \tag{3.26}$$

$$s_3 s_4 = (P_7 - P_1) \tag{3.27}$$

$$s_5 s_6 = (P_8 - P_2) \tag{3.28}$$

We convert these terms into Boolean expressions using half subtractor logical expression which can be written as:

$$s_1 = (\overline{P_6} \wedge P_0) \tag{3.29}$$

$$s_2 = (P_6 \oplus P_0) \tag{3.30}$$

$$s_3 = (\overline{P_7} \wedge P_1) \tag{3.31}$$

$$s_4 = (P_7 \oplus P_1) \tag{3.32}$$

$$s_5 = (\overline{P_8} \wedge P_2) \tag{3.33}$$

$$s_6 = (P_8 \oplus P_2) \tag{3.34}$$

The left shifting of $(P_7 - P_1)$ by one bit is expressed as follows in Boolean expressions:

$$s_7 = (s_3 \wedge s_4) \tag{3.35}$$

$$s_8 = s_4 \tag{3.36}$$

$$s_9 = 0 \tag{3.37}$$

The addition of the terms $(P_6 - P_0)$ and $(P_8 - P_2)$ are simplified as follows:

$$y_1 = (s_5 \wedge s_6 \wedge \overline{(s_1 \oplus s_2)}) \vee (s_1 \wedge s_2 \wedge \overline{s_5} \wedge \overline{s_6}) \tag{3.38}$$

$$y_2 = y_1 \vee (\overline{s_1} \wedge s_2 \wedge \overline{s_5} \wedge s_6) \tag{3.39}$$

$$y_3 = (\overline{s_1} \wedge \overline{s_2} \wedge s_6) \vee (s_2 \wedge \overline{s_5} \wedge \overline{s_6}) \tag{3.40}$$

The addition of all the terms in the y-direction are simplified as follows:

$$G_{y_1} = (\overline{(y_1 \oplus y_2)} \wedge s_7 \wedge s_8) \vee (y_1 \wedge y_2 \wedge y_3 \wedge \overline{s_4} \wedge \overline{t_7}) \tag{3.41}$$

$$G_{y_2} = G_{y_1} \vee (\overline{y_1} \wedge y_2 \wedge y_3 \wedge \overline{s_7} \wedge s_8) \tag{3.42}$$

$$G_{y_3} = (\overline{y_1} \wedge y_2 \wedge s_8) \vee (y_1 \wedge y_2 \wedge y_3 \wedge \overline{s_4} \wedge \overline{s_7}) \tag{3.43}$$

$$G_{y_4} = (\overline{(y_1 \oplus y_2)} \wedge y_3) \vee \overline{s_7} \vee s_8 \tag{3.44}$$

The final term G_y is computed by logical ORing all the terms in x-direction, i.e.,

$$|G_y| = G_{y_1} \vee G_{y_2} \vee G_{y_3} \vee G_{y_4} \tag{3.45}$$

Finally, we perform the logical OR operation to get the resultant magnitude of G as follows:

$$|G| = |G_x| \vee |G_y| \tag{3.46}$$

Equation 3.46 gives us the final edge detected pixel value at a particular location in an image.

Figure 3.5 shows the block level diagram of a 3 x 3 block of a Processing Element (PE).This process is iterated throughout all the image pixels and we get the complete edge detected raw binary image.

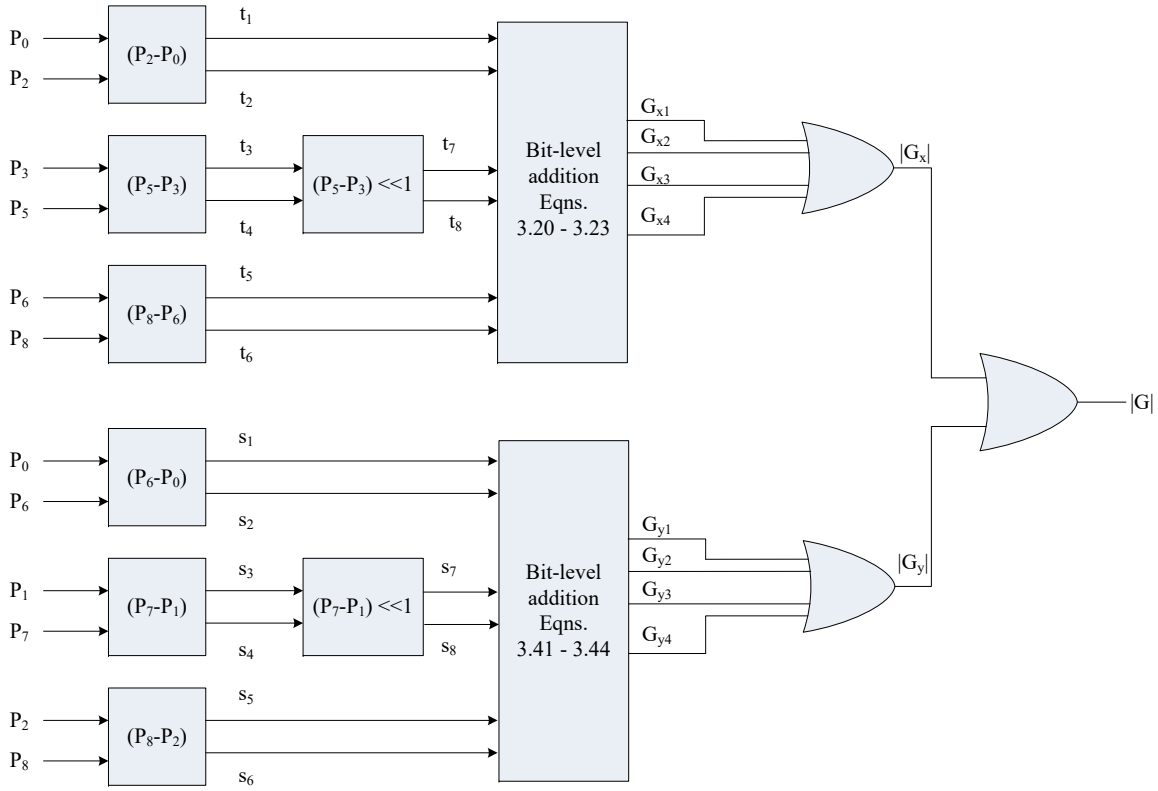

Figure 3.5: Block Level Diagram of the Proposed PE Architecture

3.2.2 An Illustrative Example

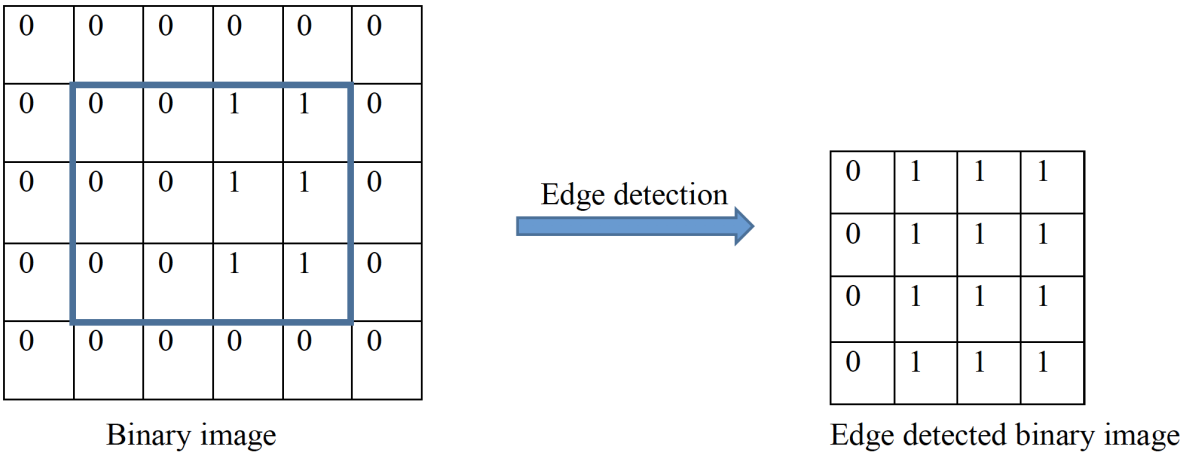

Figure 3.6: Edge Detection in an Image

Figure 3.6 shows an illustrative example of the detection of an edge from a binary raw image. The Sobel operator equations are applied to the Figure 3.6 (left-hand side) 4 x 4 matrices binary raw image with zeros padded on the outer rows and column to generate 6x6 matrices image. We will obtain the Sobel edge detected 4 x 4 matrices raw image as shown on the right-hand side.

3.3 Summary

In this Chapter, we presented in detail the proposed In-memory CMOS VLSI Bit-sliced architecture for Sobel edge detection technique. We also discussed the steps involved in obtaining the raw binary image file format. We explained the realization of the Sobel edge operator in terms of basic boolean logic gates. A simple example is presented to show the detection of an edge from a binary raw image.

CHAPTER 4: EXPERIMENTAL RESULTS

In this Chapter, we first discuss experimental flow to validate the proposed method. We also analyze the report by comparing the expected and actual results. We show the error rate produced in different images. Lastly, we present and discuss the experimental results.

4.1 Experimental Setup

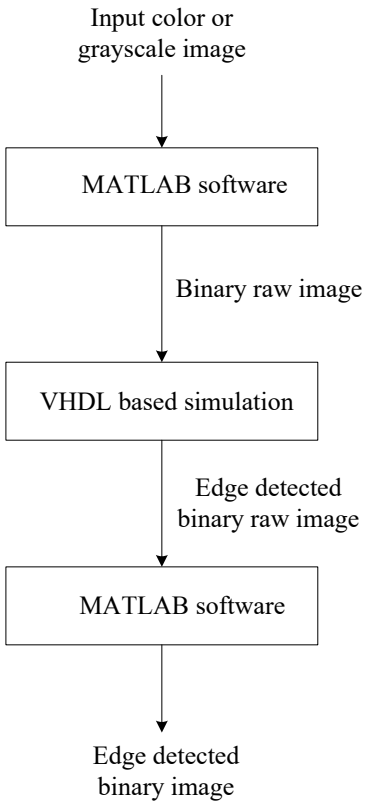

Figure 4.1: Experimental Flow

The experimental flow setup is shown in Figure 4.1. The tasks involved are as follow:

- Preparing the input image file: As we will be dealing with binary images, the first step in designing this experiment is to convert the colored or grey-scale image into a binary image. Then we have to extract the raw binary image file for manipulation pixels. We will use MATLAB software to convert the image file format. This raw binary image file will be the input to the proposed Sobel edge detection PE array.

- VHDL based simulation: Once the raw binary image is prepared, it is processed through the proposed architecture that we modeled using VHDL in Xilinx Vivado software. The architecture generates the edge detected raw binary image.

- Processing the output image: The edge detected raw binary image is processed through the MATLAB software to get the edge detected binary image in other image formats.

4.2 VHDL Modeling

We implemented a behavioral VHDL model for a 3 x 3 PE block frame. This model can output a single edge detected pixel value in an image. The complete VHDL modeling for the 3 x 3 PE block frame is shown in Subsection 4.2.1.

4.2.1 VHDL Modeling for 3 x 3 PE Block Frame

In this VHDL model, P_0, P_1, P_2, P_3, P_5, P_6, P_7, and P_8 are the neighborhood pixel values and s is the final edge detected pixel value at any point in an image. The computation of the final edge detected pixel value is performed using only logic gate operations. These logic gate operations

are simplified using operator strength reduction, bit manipulation, and common term sharing across equations. In the next section, we will discuss the validation of this VHDL model of the PE block.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity sobel_mod is
    port ( p0,p1,p2,p3,p5,p6,p7,p8 : in std_logic;
           s : out std_logic);
end sobel_mod;
architecture Behavioral of sobel_mod is
begin
    process(p0,p1,p2,p3,p5,p6,p7,p8)
    variable t1,t2,t3,t4,t5,t6, t7,t8,t9: std_logic;
    variable s1,s2,s3,s4,s5,s6, s7,s8,s9: std_logic;
    variable x1,x2,x3, y1,y2,y3: std_logic;
    variable Gx,Gx1,Gx2,Gx3,Gx4,Gy,Gy1,Gy2,Gy3,Gy4: std_logic;
    variable r1,r2,r3,r4,r5,r6,r7,r8: std_logic;
    variable r9,r10,r11,r12,r13,r14,r15,r16: std_logic;
    variable a,b,c,d1,d2,d3,d4,d5,d6,d7,d8 : std_logic;
    variable sum : std_logic;
    begin
        a := NOT P8;
```

```
t1 := ((NOT p2) AND p0);

t2 := (p2 XOR p0);

t3 := ((NOT p5) AND p3);

t4 := (p5 XOR p3);

t5 := (a AND p6);

t6 := (p8 XOR p6);

r1 := NOT t1;

r2 := NOT t2;

r4 := NOT t4;

r5 := NOT t5;

r6 := NOT t6;

s1 := ((NOT p6) AND p0);

s2 := (p6 XOR p0);

s3 := ((NOT p7) AND p1);

s4 := (p7 XOR p1);

s5 := (a AND p2);

s6 := (p8 XOR p2);

r7 := NOT s1;

r8 := NOT s2;

r10 := NOT s4;

r11 := NOT s5;

r12 := NOT s6;
```

```
t7 := (t3 AND t4);

t8 := t4;

t9 := '0';

r13 := NOT t7;

r14 := r4;

s7 := (s3 AND s4);

s8 := s4;

s9 := '0';

r15 := NOT s7;

r16 := r10;

b := (t5 AND t6 AND (t1 XNOR t2));

x1 := b OR (t1 AND t2 AND r5 AND r6);

x2 := b OR (t2 AND r5 AND ((r1 AND t6) OR (t1 AND r6)));

x3 := (r1 AND r2 AND t6) OR (t2 AND r5 AND r6);

c := (s5 AND s6 AND (s1 XNOR s2));

y1 := c OR (s1 AND s2 AND r11 AND r12);

y2 := c OR (s2 AND r11 AND ((r7 AND s6) OR (s1 AND r12)));

y3 := (r7 AND r8 AND s6) OR (s2 AND r11 AND r12);

d1 := NOT x1;

d2 := NOT x2;

d3 := NOT x3;

d4 := (x1 AND x2 AND x3 AND r13 AND r14);
```

```
        Gx1 := (t7 AND t8 AND (x1 XNOR x2)) OR d4;
        Gx2 := Gx1 OR (d1 AND x2 AND x3 AND r13 AND t8);
        Gx3 := (d1 AND d2 AND t8) OR d4;
        Gx4 := (r13 OR t8) AND (x3 AND (x1 XNOR x2));
        d5 := NOT y1;
        d6 := NOT y2;
        d7 := NOT y3;
        d8 := (y1 AND y2 AND y3 AND r15 AND r16);
        Gy1 := (s7 AND s8 AND (y1 XNOR y2)) OR d8;
        Gy2 := Gy1 OR (d5 AND y2 AND y3 AND r15 AND s8);
        Gy3 := (d5 AND d6 AND s8) OR d8;
        Gy4 := (r15 OR s8) AND (y3 AND (y1 XNOR y2));
        Gx := Gx1 OR Gx2 OR Gx3 OR Gx4;
        Gy := Gy1 OR Gy2 OR Gy3 OR Gy4;
        sum := Gx OR Gy;
        s <= sum;
    end process;
end Behavioral;
```

4.2.2 Validation of Proposed PE Block using a High Level Model

We validated the proposed VHDL model for a 3 x 3 PE block frame using a high-level model. The VHDL code represents the top module for the image array of size M x K in this

Section. The generate statement is used to instantiate PE block to compute all the edge detected pixels concurrently. Figure 4.2 shows the validation of the proposed PE block using a high-level model. In the Figure 4.2, we considered a 8 x 8 pixel image. The circles represent the pixel values in an image. The solid red square box represents a single 3 x 3 PE block frame. The green square box represents the edge detected pixel value in the image. The dashed square box represents the movement of a 3 x 3 PE block frame over the image. The 3 x 3 PE block frame is iterated over all the pixel values in the image to compute all the edge detected pixel values in the image.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
package pixel_array_pkg is
    constant K : integer := 8;
    type pixel_array is
        array (natural range <>) of std_logic_vector( 0 to K-1);
end package;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_TEXTIO.ALL;
use STD.TEXTIO.ALL;
use work.pixel_array_pkg.all;
entity sobel_NXN is
generic ( M : integer := 8 ;
```

```
            K: integer := 8;
            N : integer := 36);
    port( pixel : in  pixel_array (0 to M-1);
            s_out : out  std_logic_vector (0 to N-1));
end sobel_NXN;
architecture Behavioral of sobel_NXN is
component sobel_mod is
        port ( p0,p1,p2,p3,p5,p6,p7,p8 : in std_logic;
                s : out std_logic);
end component;
for all: sobel_mod use entity work.sobel_mod(Behavioral);
begin
    R: for i in 0 to M-3 generate
        C: for j in 0 to M-3 generate
            S: sobel_mod port map (
            pixel(i)(j), pixel(i)(j+1), pixel(i)(j+2),
            pixel(i+1)(j),              pixel(i+1)(j+2),
            pixel(i+2)(j), pixel(i+2)(j+1), pixel(i+2)(j+2),
            s_out((M-2)*i+j));
        end generate C;
    end generate R;
end Behavioral;
```

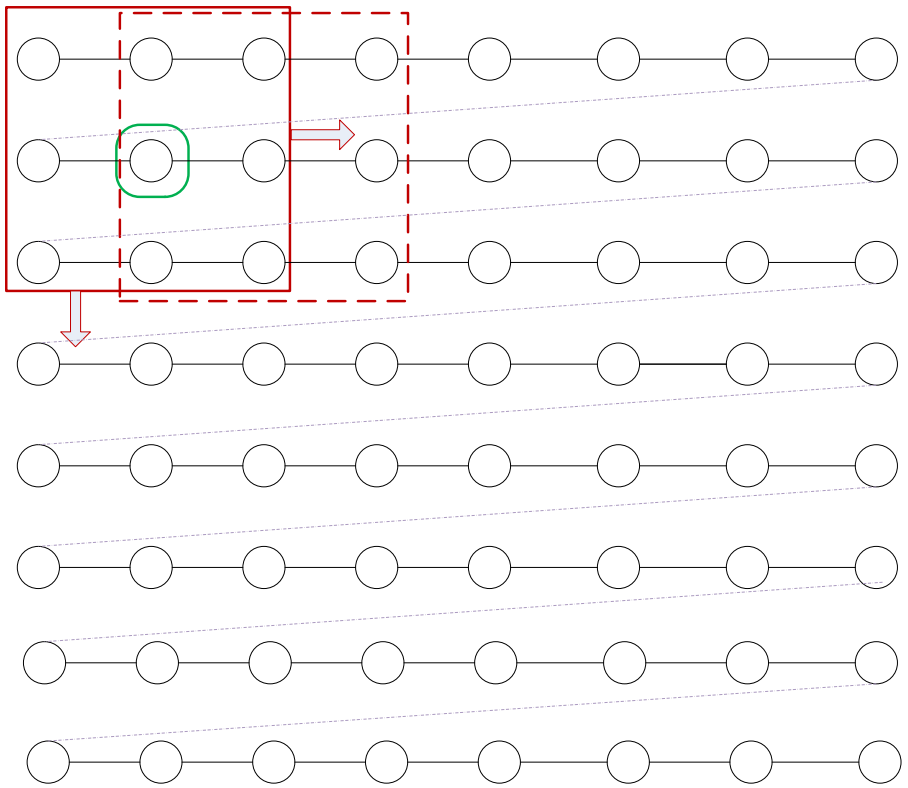

Figure 4.2: Validation of Proposed PE Block using High Level Model

4.2.3 Layout Level Implementation of PE Block

We performed the layout level implementation of a 3 x 3 PE block frame using Synopsys tool suite using 90 nm technology node. Figure 4.3 shows the layout-level implementation of a 3 x 3 PE block frame. For processing 3 x 3 PE block frame, the number of logic gates is 17 with a worst-case delay of 3.52 fs and a total bounding box layout area of 158 nm^2. The estimated average power dissipation is 0.72 μW at 0.7 V supply voltage.

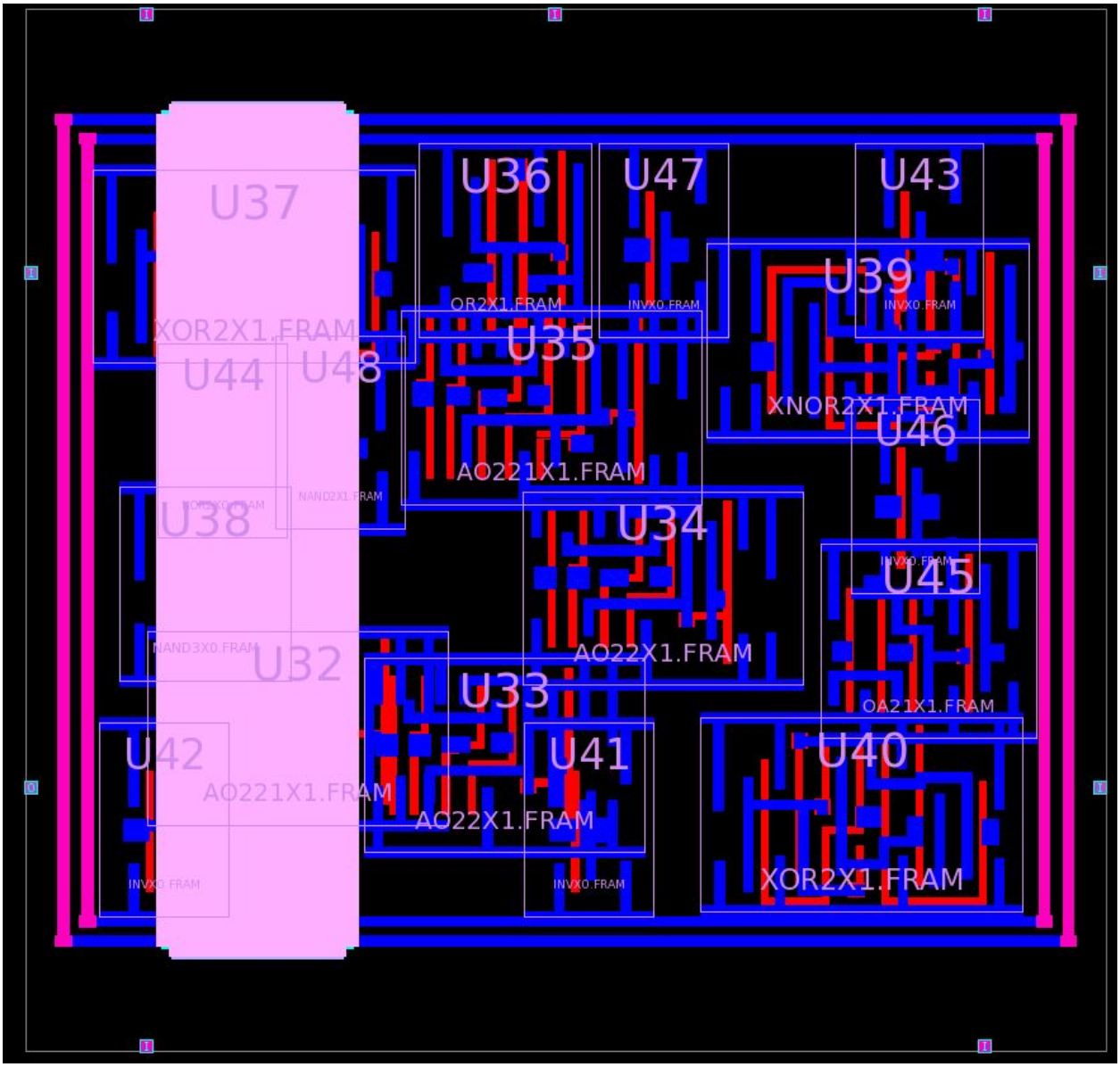

Figure 4.3: Layout Level Implementation of Proposed PE Block Implemented in 90 nm Technology

4.3 Experimental Results

We use MNIST handwritten digits (0-9) [20] and other alphabets and geometric shapes for the edge detection purpose in this experiment. The image size of MNIST data are 28 x 28 pixels and the image size of alphabets and geometric shapes are irregular which are compressed to 32 x 32

pixels. The results of this work are compared with those generated by Sobel edge detection using MATLAB software. The left-hand side of the image shows the original image, the middle image is the edge detected image generated through MATLAB software. The right-hand side image is the edge detected image generated by this proposed work.

Table 4.1: Percentage of Error

Input Image	Image Description	Total Pixels	Error in Pixels	Error %
Image1	zero	784	3	0.38
Image2	one	784	1	0.12
Image3	two	784	5	0.63
Image4	three	784	4	0.51
Image5	four	784	1	0.12
Image6	five	784	3	0.38
Image7	six	784	6	0.76
Image8	seven	784	4	0.51
Image9	eight	784	7	0.89
Image10	nine	784	7	0.89
Image11	block 1	1024	1	0.09
Image12	block 2	1024	2	0.19
Image13	cross	1024	4	0.39
Image14	P	1024	2	0.19
Image15	I	1024	2	0.19
Image16	W	1024	4	0.39

4.4 Discussion

The results are compared with the software-generated output from MATLAB software. The results are highly similar to the ones generated with MATLAB software. Even though we compressed the original images of the alphabet and geometric shapes to 32 x 32 pixel, the generated edge detected images are clear and without distortion. To check the accuracy of the edge detected images to that of the software generated image, we calculated the the number of pixels differing in the value. We report the number of differing pixels for each image in Table 4.1. We found that the error percentage

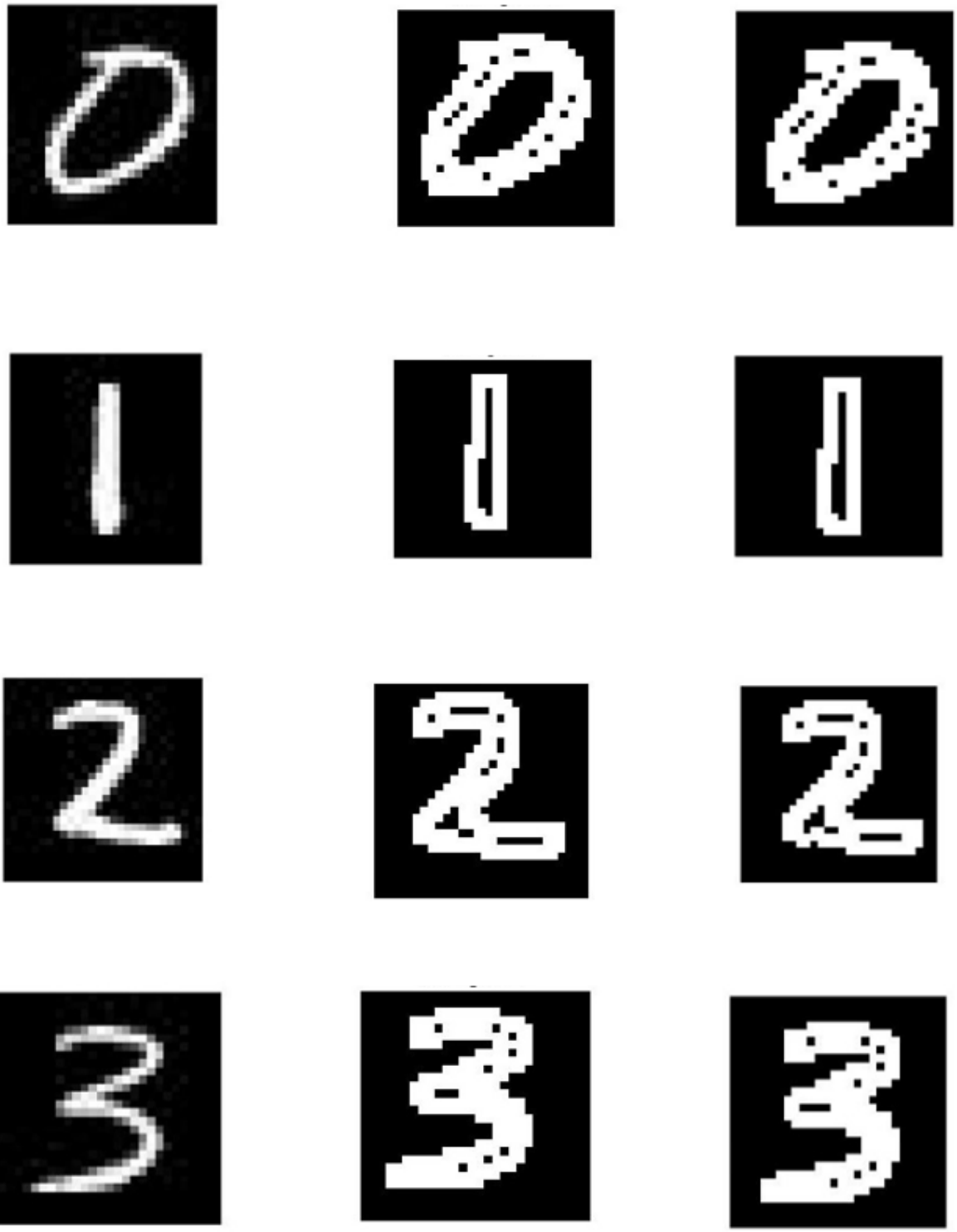

Figure 4.4: Simulation Results: Original Image (Left), Edge Detected Image through MATLAB (Middle), Edge Detected Image through Our Proposed Work (Right)

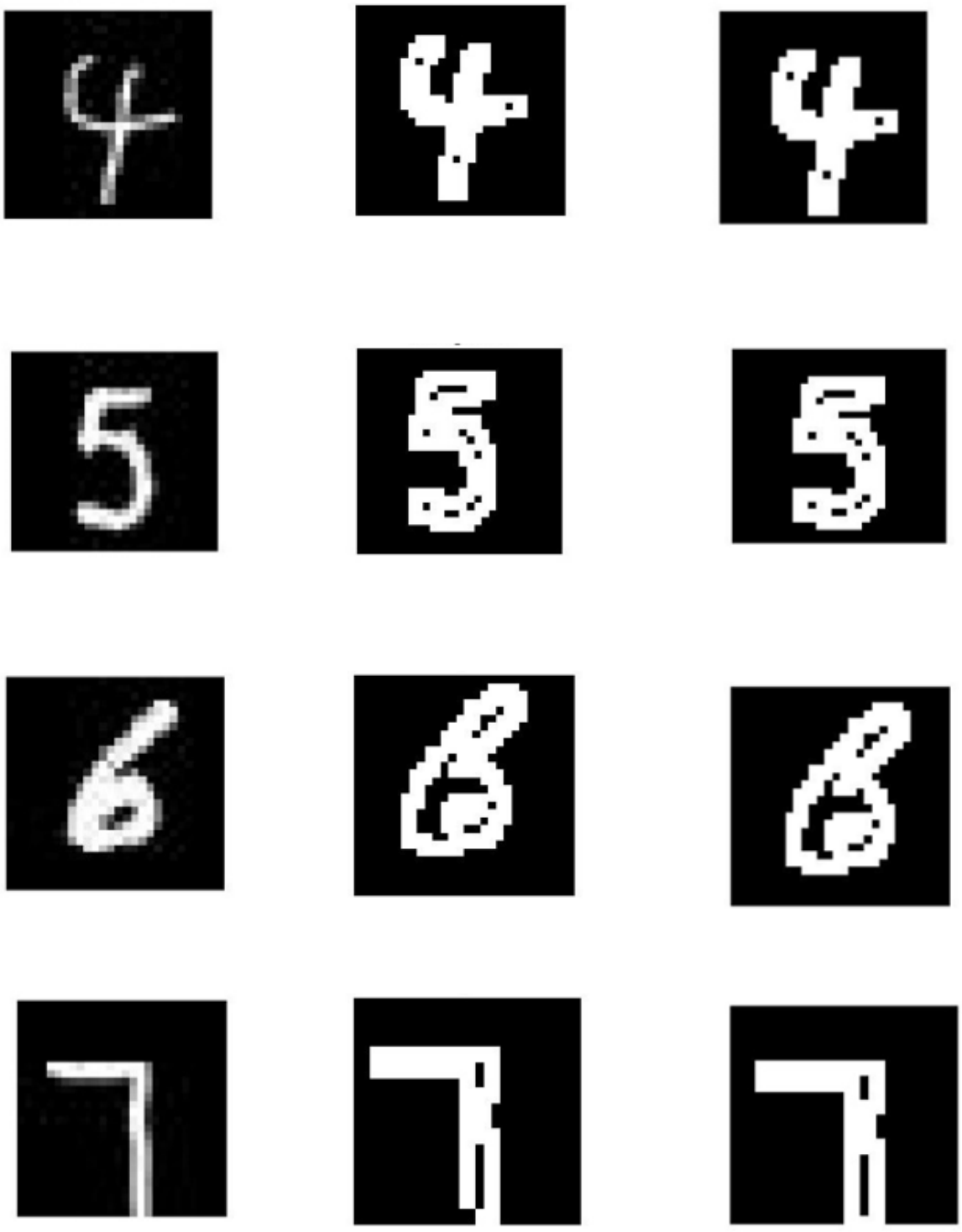

Figure 4.5: Simulation Results: Original Image (Left), Edge Detected Image through MATLAB (Middle), Edge detected Image through Our Proposed Work (Right)

Figure 4.6: Simulation Results: Original Image (Left), Edge Detected Image through MATLAB (Middle), Edge Detected Image through Our Proposed Work (Right)

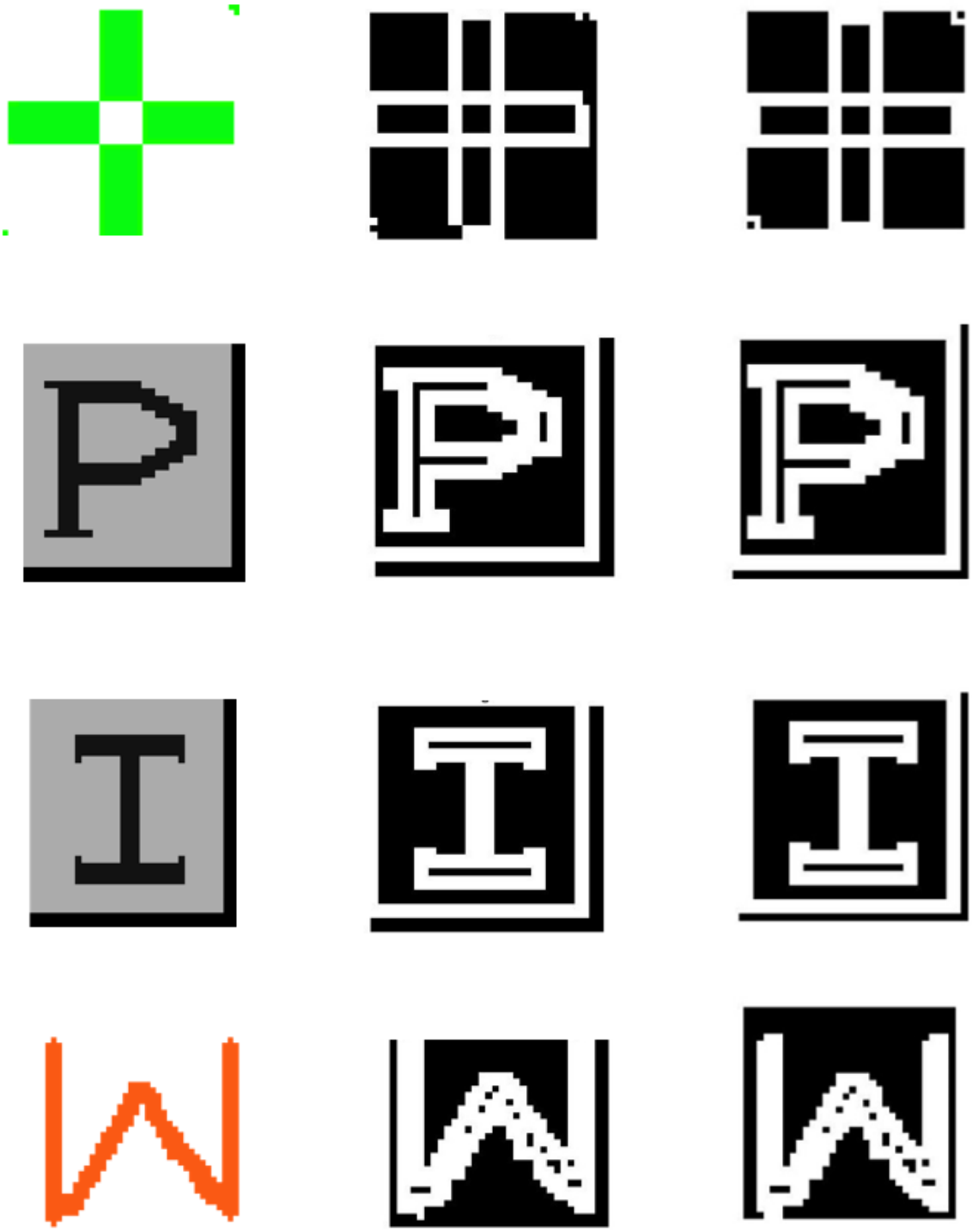

Figure 4.7: Simulation Results: Original Image (Left), Edge Detected Image through MATLAB (Middle), Edge Detected Image through Our Proposed Work (Right)

is negligible.

Table 4.2: Comparison of Various Parameters with Traditional and Proposed Approach

Attributes	[9]	[10]	[11]	Proposed work
Technology node	2 μm	1 μm	0.18 μm	90 nm
Power (Average)	430 mW	-	8.24 mW	0.18 W
Gates (512 x 512 image)	-	6,835	-	4,456,448
Image size	-	512 x 512	-	512 x 512
Core area	1,753,474 μm^2	4,580 μm^2	-	41,418 μm^2
Power supply	5V	-	1.8 V	0.7 V
Clock frequency	10 MHz	50 MHz	200 MHz	284,900 GHz (theoretical)
Worst-case delay	0.1 μs	0.02 μs	4.61 ns	3.51 fs
Energy	0.43 μJ	-	37.9 pJ	0.63 fJ
Energy- Delay Product	0.43 pJs	-	174.7 zJs	2.21 x 10^{-9} zJs

We compare the proposed model with the traditional models in terms of various design attributes such as power, gate count, area, clock frequency, worst-case delay, energy, and energy-delay product as shown in Table 4.2. We assume a 512 x 512 image as a reference for comparing these attributes. For the sake of convenience, we refresh the reader's memory by summarizing these existing approaches (for more details, see Chapter 2).

Kanopoulos, Vasanthavada, and Baker [9] proposed a design and implementation of the Sobel edge detection in CMOS technology on a single chip. The design is implemented using 2 μm CMOS technology node operated at 10 MHz clock frequency. The design architecture is highly pipelined. The area of the design is 1,753,474 μm^2 with an average power dissipation of 430 mW and worst-case delay of 0.1 μs. The energy consumption is 0.43 μJ with energy delay product of 0.43 pJs. The number of gates and image size are not specified in this model.

Boo, Antelo, and Bruguera [10] proposed the implementation of the Sobel operators using Application Specific Integrated Circuit (ASIC). The systolic processors arrays are utilized for the efficient architectural design. The design is implemented in 1 μm CMOS technology node. For

processing 512 x 512 image, the number of gates required is 6,835 with a worst-case delay of 0.02 μs and clock frequency of 50 MHz. The area of the design is 4,580 μm^2. The frequency of operation is 50 MHz. The parameters such as power, energy, and energy-delay product are not mentioned.

Kazakova, Margala, and Durdle [11] proposed a low power CMOS based design and implementation of Sobel edge detection. The design is implemented in 0.18 μm CMOS technology node. Wallace compression tree and carry-select adders are used to design the Sobel processor for computing the gradients. The average power dissipation in the design is 8.24 mW and worst-case delay is 4.61 ns. The operating clock frequency of the design is 200 MHz. The energy consumption is 37.9 pJ with energy delay product of 174.7 zJs. The image size, number of gates, and area are not mentioned by the authors.

The proposed work is implemented in 90 nm CMOS technology node. For processing a 512 x 512 image by the proposed in-memory computing architecture, the number of gates required is 4,456,448 with a worst-case delay of 3.51 fs and with a layout bounding box area of 41,418 μm^2. For processing one block frame (3 x 3 pixel block), the average power consumption is 0.72 μW. We estimated the power consumption for processing 512 x 512 image by multiplying the power consumed by one block frame and total number of pixels in 512 x 512 image. The estimated average power dissipation is 0.18 W with energy of 0.63 fJ and energy-delay product of 2.21 x 10^{-9} zJs. The theoretical clock frequency of operation to process one frame (512 x 512) is 284,900 GHz (= 1/worst-case delay). Although this much high clock frequency is not feasible, we can clock the circuit at a rate of 1-2 GHz to achieve the significant improvement in performance.

As we can see, the power consumption in the proposed work is higher than the other traditional models, this is because of the higher frequency of operation and increment in number of

gates required for processing all 3 x 3 blocks of an image simultaneously. However, the results show that there is a significant improvement in average energy, energy-delay product, and the worst-case delay of the proposed model as compared to traditional approaches even though there is an increase in area and the number of gates with the increase in image size.

The main advantage of this proposed model is that one image can be processed in constant time irrespective of the image size.

4.5 Summary

In this Chapter, we presented the experimental set up and experimental flow that is used for validating the proposed in-memory CMOS VLSI based architecture for Sobel edge detection. Expected and actual results are presented and compared. The expected result as compared with the actual result are reported using error rate. We also compared the proposed work with the other works in terms of various parameters such as power, gate, area, worst-case delay, energy, and energy-delay product.

CHAPTER 5: CONCLUSIONS AND FUTURE WORK

We conclude from this work that the proposed In-memory CMOS VLSI bit-sliced 2D architecture for Sobel edge detection for IoT image applications is an effective architectural design. The results obtained are very much promising. The way the architecture works at the bit level makes it very efficient in terms of performance and power and energy which is a requirement for IoT connected node devices. The architecture is simple in design and highly modular. Currently, the architecture is tested with binary images and can be further extended to the processing of gray-scale and color images. We can extend this work for different image filters which would be very useful for image processing applications in IoT and other application domains.

REFERENCES

[1] Statista Research Department. IoT: Number of Connected Devices Worldwide 2015-2025, Nov 2016.

[2] D. Evans. The Internet of Things: How the Next Evolution of the Internet is Changing Everything. *CISCO white paper*, 1(2011):1–11, 2011.

[3] R. C. Gonzalez and R. E. Woods. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., USA, 2006.

[4] R. Jain, R. Kasturi, and B. G. Schunck. *Machine Vision. Volume 5*. McGraw-Hill New York, 1995.

[5] L. G. Roberts. *Machine Perception of Three-dimensional Solids*. PhD thesis, Massachusetts Institute of Technology, 1963.

[6] L. S. Davis. A Survey of Edge Detection Techniques. *Computer Graphics and Image Processing*, 4(3):248–270, 1975.

[7] J. M. S. Prewitt. Object Enhancement and Extraction. *Picture Processing and Psychopictorics*, 10(1):15–19, 1970.

[8] I. Sobel. An Isotropic 3 x 3 Image Gradient Operator. *Presentation at Stanford A.I. Project 1968*, 2014.

[9] N. Kanopoulos, N. Vasanthavada, and R. L. Baker. Design of an Image Edge Detection Filter using the Sobel Operator. *IEEE Journal of Solid-State Circuits*, 23(2):358–367, April 1988.

[10] M. Boo, E. Antelo, and J. D. Bruguera. VLSI Implementation of an Edge Detector Based on Sobel Operator. In *Proceedings of Twentieth Euromicro Conference. System Architecture and Integration*, pages 506–512, Sep. 1994.

[11] N. Kazakova, M. Margala, and N. G. Durdle. Sobel Edge Detection Processor for a Real-time Volume Rendering System. In *2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No.04CH37512)*, volume 2, pages II–913, May 2004.

[12] D. Hanes, G. Salgueiro, P. Grossetete, R. Barton, and J. Henry. *IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Things.* Cisco Press, 1st edition, 2017.

[13] A. B. Dorothy, S. B. R. Kumar, and J. J. Sharmila. IoT Based Home Security through Digital Image Processing Algorithms. In *2017 World Congress on Computing and Communication Technologies (WCCCT)*, pages 20–23, Feb 2017.

[14] H. T. Tseng, H. G. Hwang, W. Y. Hsu, P. C. Chou, and I. C. Chang. IoT-Based Image Recognition System for Smart Home-Delivered Meal Services. *Symmetry*, 9(7):125, Jul 2017.

[15] A. Frank, Y. S. Khamis Al Aamri, and A. Zayegh. IoT Based Smart Traffic Density Control using Image Processing. In *2019 4th MEC International Conference on Big Data and Smart City (ICBDSC)*, pages 1–4, Jan 2019.

[16] A. Kapoor, S. I. Bhat, S. Shidnal, and A. Mehra. Implementation of IoT (Internet of Things) and Image Processing in Smart Agriculture. In *2016 International Conference on Computation*

System and Information Technology for Sustainable Solutions (CSITSS), pages 21–26, Oct 2016.

[17] S. Rane, A. Dubey, and T. Parida. Design of IoT Based Intelligent Parking System using Image Processing Algorithms. In *2017 International Conference on Computing Methodologies and Communication (ICCMC)*, pages 1049–1053, July 2017.

[18] W. A. Wulf and S. A. McKee. Hitting the Memory Wall: Implications of the Obvious. *ACM SIGARCH Computer Architecture News*, 23(1):20–24, 1995.

[19] The USC-SIPI Image Database, [Online]. Available: http://sipi.usc.edu/database.

[20] Y. LeCun. The MNIST Database of Handwritten Digits. *http://yann. lecun. com/exdb/mnist/*.