A CMOS Imaging Device for Visual Prosthetics Using On-Pixel Gray-Scale Erosion for Edge Detection

by

Duha Jabakhanji

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Master of Applied Science

Ottawa-Carleton Institute for Electrical Engineering
Department of Electronics
Faculty of Engineering
Carleton University
Ottawa, Canada

Your file Votre référence
ISBN: 978-0-494-26992-3
Our file Notre référence
ISBN: 978-0-494-26992-3

Abstract

This thesis proposes a new design for on-pixel implementation of gray-scale morphology, aimed at visual prosthetics. These implants rely on providing the patient with a sketch or a realistic edge representation of the real world. Edge detection can be carried out using classical algorithms or mathematical morphology.

Threshold decomposition of gray-scale images allows for the simplification of gray-scale mathematical morphology into binary morphology. Binary morphology can be implemented on pixel by using AND, OR and inverter gates. Gray-scale morphology on the other hand, needs more complex operations. By using threshold decomposition, these operations can be carried out in terms of binary morphology. To carry out threshold decomposition in hardware, a specialized system is needed to decompose the gray-scale image into M binary images, where M is the largest value a pixel can have in the gray-scale image.

This thesis proposes a new method for gray-scale mathematical morphology, called bitwise decomposition of gray-scale images. The proposed method reduces the number of binary images produced from M to N, where N is the number of bits representing the pixels of a gray-scale image. Binary erosion can then be carried out on the generated binary images, and the eroded gray-scale image reconstructed.

Erosion was chosen for edge detection since it produces correctly placed edges in the edge image. Once the gray-scale eroded image is available, it is used in morphological edge detection, where the results are comparable with classical algorithms. Once testing the new method in software is complete, the method is implemented in hardware.

A pixel employing the method of bitwise decomposition is designed in 0.13 μm technology. The pixel is then placed in three array structures, a 4x4 , 8x8 and a16x16 array. All arrays produced correct results as expected.

Layout of the pixel is drawn and tested using the layout-level simulations. The results match that of schematic-level tests. The fill factor of the pixel came to 22 % and the device has a dynamic range of 46 dB. Pixel dimension is 17x17 μ m, and transistor count 107. An analog to digital converters resolution is 8-bits and has a conversion time of 40 μsec. The power dissipation of the 16x16 is 0.38 mW.

Acknowledgements

I would like to express my sincere thanks to my supervisor, Dr. Maitham Shams, for his continuous support, guidance and encouragement. Many thanks also go to Dr. John Knight for his valuable help and involvement during the course of this work. The help provided by the department staff is greatly appreciated.

The inspiration to carry out my master's thesis came from my parents, Dr. Marwan Jabakhanji and Dr. Marwa Wakeel. I would like to thank them for their support, prayers and continuous encouragement. This work would not have been completed without my husband, Osman Sakalla's, valuable support, continuous prayers and encouragement. He has always been there for me. The inspiration for my thesis topic came from my best friend Ahlem Khaitous, I appreciate her support and inspirational talks. Special thanks are extended to my colleagues and the VLSI group for their valuable feedback and help.

"Behold! in the creation of the heavens and the earth, and the alternation of night and day,- there are indeed Signs for men of understanding,-" (Quran- 3:190)

To My Dear Husband, my Family and Friends

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ADC Analog to Digital Conversion

AMS Analog Mixed Signal

APS Active Pixel Sensor

DPS Digital Pixel Sensor

DR Dynamic Range

FF Fill Factor

FPN Fixed Pattern Noise

FSM Finite State Machine

PE Processing Element

PPS Passive Pixel Sensor

PTL Pass Transistor Logic

SE Structuring Element

SNR Signal to Noise Ratio

T Temperature in Kelvin

TS Tristate

VCI Visual Cortical Implants

Chapter 1

Introduction

1.1 Thesis Motivation

Sight is considered the most precious of senses, the eye is a jewel. Millions of people worldwide lose their eyesight due to a number of diseases or injury. With the modern advancement in technology, partial restoration of sight is possible. Today, there are several kinds of implants available for the blind: subretinal implant, epiretinal implant, optic nerve implant and Visual cortical implants (VCI), all of which are still being tested. Of these implants, the most practical one is the visual cortical implant. This is due to its ability to restore vision for patients with a wide variety of blindness, where the visual cortex is intact.

Imaging devices have been modified to serve as implants, imitating the retina as closely as possible. "Artificial Retinas" are composed of a light sensing device and a processing element. This emerging technology, which falls under complementary metal-oxide semiconductors (CMOS) imagers is gaining popularity in many applications, one of which is visual prosthetics.

The main drawback of CMOS imagers is the low fill factor (FF). Fill factor is the percentage area of the pixel that is photosensitive. A lower fill factor means a noisier image, with less image resolution. A low fill factor means a larger imaging device for a given number of pixels. For epiretinal implants, the imaging device should be as small as possible. Many digital pixel sensors (DPS) carry out partial image processing on-pixel. These pixels provide massively parallel image reads and allow for real time application.

Edge images are the most useful to blind patients. Edge detection algorithms are

computationally intensive and cannot be implemented on-pixel. A representation of the real world, or a sketch, is useful for an epiretinal or a visual cortical implant. Thus an imaging device that address these needs is required.

This imaging device would need to be portable, and able to give a representation of the real world to the patient without the need for external processors. This stand alone unit should be implantable or mounted on a pair of glasses, depending on the need.

1.2 Overview of Previous Work

There have been several designs published, both for visual prosthetics and machine vision, which will be described briefly in this section. Table 1.1 summarizes the different designs with on-pixel analog-to-digital converter (ADC) but without a processing element. A processing element within a pixel is usually composed of an arithmetic-logic-unit (ALU), gates or any circuitry that will processes the signal. The dynamic range (DR) in Table Table 1.1 refers to he sensors' ability to image a scene in a range of illuminations.

Table 1.1: Digital pixels with no processing element

Ref	Technology	FF	Pixel Size	DR	PD	Speed	Use
[1]	0.18μm	21%	16x16μm	NA	5W/m^2	100MHz	VCI
[2]	0.18μm	29%	13.8x13.8μm	96dB	5W/m^2	50MHz	VCI
[3]	0.18μm	25%	9.4x9.4μm	NA	50mW	167MHz	General
[4]	0.35μm	12%	45x45μm	85dB	5.28μW	25MHz	General
[5]	0.35μm	NA	NA	100dB	NA	100KHz	General
[6]	0.35μm	20%	50x50μm	90dB	10mW	NA	General
[7]	0.6μm	14%	32x30μm	48.9dB	0.1W/cm^2	8MHz	General
[8]	0.35μm	24-41%	75x78μm	NA	20mW	5KHz	Subretinal Implants

In [1] and [2], by Trepanier et al. the pixel is a simple photodiode, reset circuitry, comparator and an 8-Bit DRAM. The comparator triggers the write signal of the DRAM by comparing the photodiode voltage with an analog ramp. Once disabled, the write signal stores the value fed to the memory from a gray code counter. This method of on-pixel ADC is widely employed in pixels.

Similarly, [3], [9] and [10] by Kleinfelder et al, use the same on pixel ADC method. The reset circuitry and comparator however, differ from those used by Sawan. In this paper, a

photogate was used reducing the fill factor from 25% for a photodiode to 15%.

Using the same on-pixel ADC principle, Kitchen et al. propose an asynchronous self-resetting pixel [4]. The asynchronous scheme uses signals generated by the comparator to reset the circuitry. Although this design delivers a higher dynamic range (DR) as stated in [11], it comes at a cost of increased circuitry, and reducing the fill factor (FF).

In an attempt to increase the dynamic range of a CMOS sensor, Bermak and Kitchen propose an adaptive logarithmic DPS in [5]. Their pixel uses a similar ADC approach to those proposed in [3], but rather than having the quantization steps fixed, it uses a feedback circuit to determine the step size.

A similar design is presented in [6] by Bermak and Yung. This pixel also uses a nonuniform quantization scheme for ADC to increase the dynamic range of the sensor. In this design, the sensor can operate in high 8-bit mode resolution, or low 4-bit mode resolution. This is achieved by having each pixel contain two photodiodes. In the 8-bit mode, one photodiode circuitry is switched off, where as in the 4-bit mode, both photodiodes are on, but the dynamic random access memory (DRAM) is split between the two.

On a different note, the design in [7] by Culurciello et al. attempts to mimic the human retina in its signaling system. Instead of representing each pixel by an 8-bit value, it generates spikes or pulses to encode the light intensity. This design is also asynchronous in its signaling scheme. The added circuitry allows the pixels to access the bus asynchronously at the cost of a lower FF.

The pulse generation principle was also exposed in [8] by Mazza et al, but with a simpler design. The paper describes two designs for generating the pulses. The simplicity of this design comes at the cost of lower speeds and thus lower DR. Although the DR was not stated, it can be inferred that it is low from the speed.

All the above pixels encoded the photodiode voltage but did not carry out any signal processing at the pixel level. Since the retina carries out signal processing at the photoreceptor level, several designs have attempted to mimic that feature. Incorporating on-pixel processing elements into the the design allows the chip to carry out signal processing. These chips fall under what is called "artificial retinas" which have many advantages. Some of the advantages include parallel processing and windowing, which will be discussed in more detail in Chapter Two. Table 1.2 summarizes the literature on pixels with a processing

element.

Table 1.2: Pixels with a processing element in the literature

Ref	Technology	FF	Pixel Size	DR	PD	Speed	Use	Type
[12]	0.35μ	10%	67.4x67.4μm	NA	NA	10MHz	AI	Digital GS
[13]	0.6μ	10%	120x120μm	NA	500μW	200MHz	General	Digital Binary
[14]	0.5μ	15%	29x29μm	NA	78μW	150KHz	General	Analog
[15]	0.18μ	NA	NA	NA	NA	NA	General	Analog
[16]	0.5μ	15%	80x80μm	NA	50mW	1KHz	General	Analog

The chip implemented by Kagami et al. is covered by many papers discussing the entire chip, including the pixel array and the controller which are presented in [12] [17] [18] [19] [20]. The chip aims at providing as many processing functions on pixel as possible. Each pixel contains a photodiode, reset circuitry, comparator, 24-bit DRAM, a full adder, several multiplexers and latches. The multiplexers, latches and full adder make up the processing element. The full adder allows the pixels to carry out several functions such as morphology on binary images and edge detection on a 6-bit gray-scale image. This versatility comes at the price of a much lower fill factor. It also carries out much of its image processing on binary images rather than gray-scale.

Another digital pixel sensor is the one discussed in [13] by Fey et al. This chip aims at carrying out binary morphology at the pixel level. Each pixel contains several multiplexers, AND and XOR gates. The main drawback of this chip is its fill factor and the use of binary rather than gray-scale images, where binary images lose many of the details in a scene.

Several analog designs have attempted to incorporate processing elements into the pixels, and carry out analog arithmetic on the signals. In [15], by Zhang and Wang, a new edge detection algorithm is proposed. The algorithm aims at reducing the number of connections each pixel and its neighbours need. By reducing the need to connect eight neighbours to each pixel down to four, the algorithm simplifies edge detection, making it easier to implement in hardware. The algorithm takes the values from the pixels in its mask, then carries out subtraction. The results absolute value is taken then compared with a threshold, generating a binary signal. Thus the binary values represent the gradient of the pixel in the direction of the mask, where a large change in gradient indicates an edge. The algorithm is implemented using a series of current comparators and subtractors.

Zhang and Wang only present the algorithm and some schematic-level simulation results. The fill factor, pixel size, dynamic range, and other values are not mentioned in the paper.

The last paper we discuss is [16] by Barbaro et al. Their design aims at finding the magnitude and direction of spatial gradients of the image. Each pixel carries out analog signal processing using a multiplier. The pixels access two asynchronous buses, one for the magnitude, and another for the gradient. The main drawback is that the pixel arrays need external bias circuitry, digital control, and communications circuitry, adding extra external circuitry circuitry and the need for more connections.

Review of the above-mentioned literature on digital pixel sensors have aided in setting the objectives, outlined next, for this thesis.

1.3 Objectives of the Thesis

The objectives of this thesis centre around designing and implementing a CMOS imaging device for visual prosthetics. This device, unlike others, will carry out the sole operation of gray-scale edge detection. These objectives are as follows:

- Design an CMOS imaging device aimed at visual prosthetics
- The imaging device should provide a realistic sketch or representation of the real world for the patient
- Increase the fill factor of the pixels by decreasing the size of the processing element
- Include morphological edge detection into the pixel design
- Keep power consumption at a minimum without compromising reliability

The thesis aims at realizing the objectives by implementing a new pixel design. This design utilizes gray-scale mathematical morphology to carry out edge detection in a compact format.

1.4 Thesis Organization

The organization of this thesis is as follows: Chapter 2 provides the reader with a brief description of the human visual system, covering the eye, the retina and the visual cortex. The Chapter then moves on to briefly describe the different types of visual prosthetics, with a more detailed description of visual cortical implants and their components.

Once a basic understanding of visual cortical implants is established, the thesis describes the different types of imaging devices available. Chapter 2 will conclude with the basics of CCD and CMOS imagers and how they apply to artificial retinas.

Then Chapter 3 gives the reader an overview of edge detection, using popular algorithms and mathematical morphology. This chapter will cover how these methods can be integrated into the artificial retina.

The first three chapters are aimed at giving a thorough background for the thesis topic. Thus Chapter 4 will cover the design of our compact artificial retina for visual cortical implant using gray-scale mathematical morphology for edge detection. It will then move on to describe the layout details of the artificial retina. The last section of the chapter will cover the test bench results of the design. Finally, Chapter 6 will conclude the thesis and give a discussion of future work.

Chapter 2

Background into Human Vision, Visual Prosthetics and Imaging Technology

2.1 Human Vision and the Retina

2.1.1 Introduction

When attempting to mimic a physiological system, whether for a prosthetic or artificial intelligence, a thorough understanding of the biological system is needed. This chapter presents a brief overview of the human visual system and more details about the retina. The discussion starts with an overview of the human eye, then moves on to describing the functionality of the retina, ending with a quick overview of the visual cortex.

2.1.2 The Human Eye

The eye is composed of two spheres: a small portion of a strongly curved sphere making up the cornea and a large part of a not so strongly curved sphere. The adult eye measures about 2.5 cm in diameter. It is located at the front of the skull in a protected cone-shaped cavity called the orbit or the eye socket. The eye is surrounded by fatty connective tissue that provides protection and lubrication for movement [21] [22].

As seen in Figure 2.1, a cross section of the eye reveals it to be a complex organ. The sclera is the tough white outer coating that provides strength, structure and protection. The sclera is transparent at the front of the eye, forming the cornea and providing "the

window of the eye". The cornea allows light to pass through and is responsible for the majority of the light bending and focusing [21] [22].

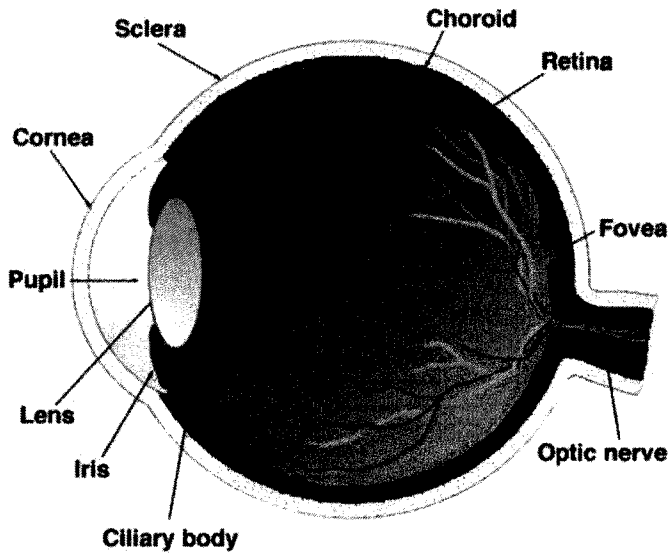


Figure 2.1: Sagittal view of the eye [23]

The iris, the coloured part of the eye, lies beneath the cornea. The iris muscles contract and relax to make the pupil bigger or smaller. The pupil is the black part in the middle of the iris. It acts as the aperture of the eye and allows more or less light to enter the eye. It is black so because light is absorbed within the eye. The lens lies beneath the iris and adjusts the focus of the light on the retina. It becomes thicker and thinner by the contractions of muscle fibers connected to the lens [21] [22]. The Vitreous is a jell-like substance that fills the eye. It provides the spherical shape and gives support [21] [22].

The retina is the inner most layer in the eye. It receives the light and processes it into electrical signals that are sent to the optic nerve and then to the brain. The retina carries out several signal-processing steps on the light. It provides edge detection, light intensity and colour information. On the retina, near the optic nerve, is the macula. The macula has a high concentration of light-sensitive rods and cones that have a one-to-one association with the signal processing nerve cells. Thus, the macula provides fine images

that are used for fine and intricate visual tasks. Once the retina processes the image, it sends the image in the form of electrical signals through the optic nerve to the brain for further processing and interpretation [21] [22].

2.1.3 Anatomy of the Retina

Only 0.5 mm thick, the retina lines the back of the eye and provides the first stages of sight. Figure 2.2 illustrates that, at the very back of the retina, the farthest away from the Vitreous, are the rods and cones. The rods and cones are the photo sensors of the eye. Light must travel through the entire thickness of the retina before reaching the rods and cones. Once stimulated, the rods and cones translate light into first a biochemical signal then into an electrical signal. The rods are responsible for gray scale vision and work under dim light. There are three different kinds of cones: short, medium and long wavelength cones, which are responsible for colour vision and work under bright light [23] [24] [25].

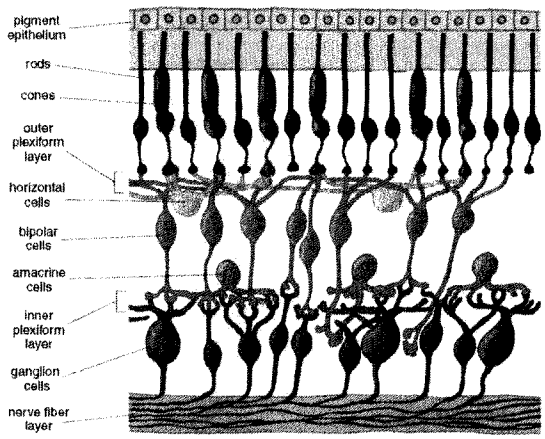


Figure 2.2: Layers of the retina [23]

The electrical signals from the rods are then sent to the horizontal and bipolar cells. These cells carry out signal processing functions on the electrical signal. The signal is then passed on to the amacrine cells for further processing and, finally, to the ganglion cells whose axons lead to the optic nerve [23] [24] [25].

There are two types of bipolar cells. "On" bipolar cells produce a signal when they receive a signal from photodetectors. "On" cells detect light on dark backgrounds. "Off"

bipolar cells, on the other hand, produce a signal when there is no signal from the photoreceptors. The latter detect dark on light backgrounds. A bipolar cell can also differentiate signals sent to its centre from those sent to its periphery. With the help of horizontal cells, edges can be detected by comparing neighbouring signals [24].

Horizontal cells not only provide signals to bipolar cells for image refinement, they also provide feedback to the photoreceptors, adjusting their sensitivity. The main function of the ganglion cells is to encode the signals from the amacrine, horizontal and bipolar cells. Some ganglion cells are also photo-receptive and have an off centre and on centre detection method to further refine the image processing. There are approximately 130 million photoreceptors and about one million ganglion cells, giving the retina a very high degree of convergence, a lower photoreceptor to ganglion ratio, this gives rise to less visual acuity, and thus peripheral vision. At the macula, there is a one to one correspondence between photoreceptors and ganglion cells, providing the high degree of detail [23] [24] [25].

One of the most important operations taking place in the retina is lateral inhibition. Lateral inhibition occurs in a network when a positive outcome in one of its elements induces a negative outcome in its neighbours. This occurs when a nerve cell responds to light reaching the centre of its receptive field is the opposite of that of light in the surrounding cells. Lateral inhibition is used in the retina for simple edge enhancement and hue and colour discrimination [23].

2.1.4 The Visual Cortex

The visual cortex is folded just like the brain, where peripheral vision is hidden in the folds, and fovea vision is located on the surface [26]. It is organized in layers, where each layer has a certain orientation that is responsible for image processing. The functionality of the layers are well understood, where the orientations provide the signal processing power [23]. It has also been proven that the visual cortex holds a form of temporary memory. This memory holds the images seen between processing steps [27].

2.1.5 Conclusion

Despite advances in technology, the human visual system has not been fully understood. Vision includes a complex system of connected neurons that carry out complex processes.

What has been established however, is the functionality of the retina. The retina perceives an image by extracting light intensities, colour and edges. The signals from the retina are compacted to allow extreme details in the centre of vision, but much less details in the periphery.

2.2 Types of Visual Prosthetics

This section briefly describes the different types of prosthetics being researched today. The section introduces subretinal, epiretinal and optic nerve implants. It then goes into more detail on visual cortical implants and finally describes parameter of electrical stimulation, which apply for all the implants described. Each implant has its advantages and disadvantages, their uses also differ depending on the type of blindness.

2.2.1 Subretinal Implants

The subretinal implant is the most successful of the vision implants. Patients with age-related macular degeneration and retinitis pigmentosa have degraded photoreceptors, but the remainder of the retina is intact. This means that the horizontal, bipolar and amacrine cells are still viable. Subretinal implant aims at replacing the photoreceptors in the retina. It is implanted in the subretinal space, where the damaged photoreceptors are found, between the retina and the pigment epithelium. Figure 2.3 depicts a typical subretinal implant [28] [29].

The subretinal implant consists of a photodiode array, circuitry to generate pulses that would stimulate the retina, and an electrode array. The chip needs no external power supply, it has a solar array that is powered by the incident light entering the eye [28] [29].

Subretinal implants are simple and form a stand-alone unit. Light entering the eye stimulates the photodiode array and induces the solar array to provide power. The photodiode array then drives the pulse generation circuitry. This circuit provides pulses that are proportional to the intensity of the light. The more intense the light, the higher the frequency of the pulses. The pulses are then sent to the electrode array which stimulates the bipolar and horizontal cells [28] [29].

The disadvantage to the subretinal implant is its remote location in the subretinal space.

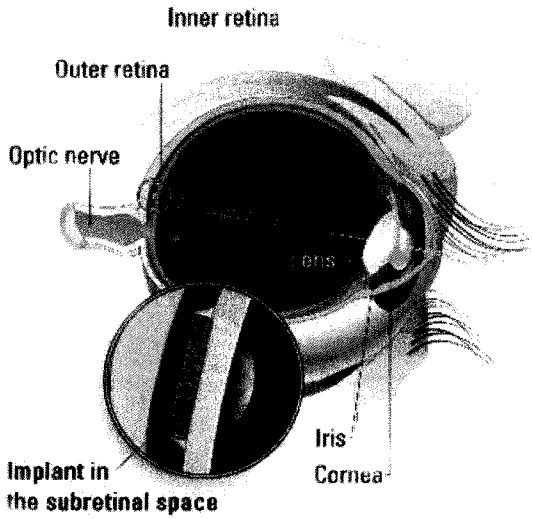


Figure 2.3: Subretinal implant and its location in the retina [30]

This makes it hard to supply external power,if not using a solar array, or to monitor the device if needed. Also, the implant comes in contact with retinal cells and tissues, which could cause damage furthermore, the subretinal space does not provide a means to dissipate the heat generated by the implant. This type of implant is also the most limited prosthetic, where the patient has to have most of the retina intact [29].

2.2.2 Epiretinal Implants

To place an epiretinal implant, the ganglion cells have to be intact. Figure 2.4 shows a typical epiretinal implant along with its external support system. Since the epiretinal implant requires the ganglion cells for stimulation, this implant works for patients with age-related macular degeneration and retinitis pigmentosa [29].

The epiretinal implant consists of an intraocular unit and extraocular unit. The extraocular unit consists of a video camera, a video processor, a telemetry encoder chip, a radio frequency amplifier,'all mounted on a pair of glasses, and a primary induction coil mounted on the eye. The glass lenses provide a link to a battery powering the system. The camera captures the image, the processor performs edge extraction and other image processing tasks. The image is translated into pixels sent to a chip implanted in the eye.

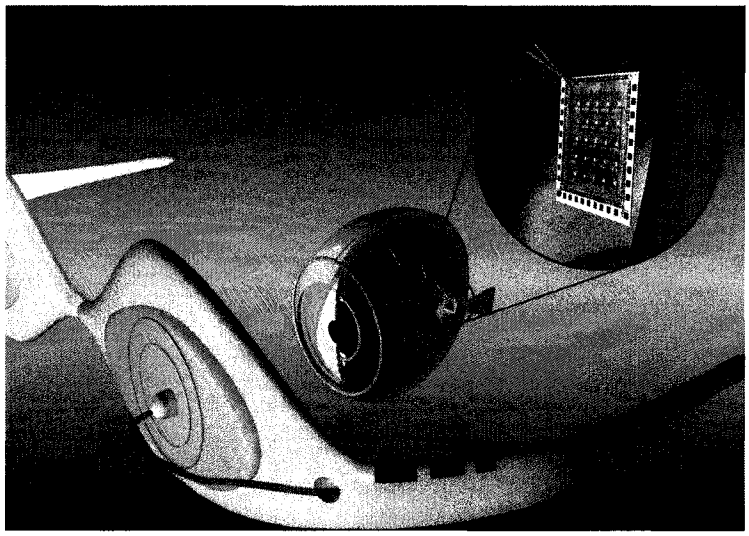


Figure 2.4: Epiretinal implants [29]

Power is transferred to the implant via a primary coil, which has been proven to be inefficient. An infrared laser can be used instead, but that poses a threat of damaging the eye if the laser is misaligned [28] [29].

The intraocular unit consists of a secondary coil, or a photovoltaic receiver to capture the laser, a receiver and regulator, a retinal simulator with a telemetry protocol decoder, a simulating signal array and an electrode array. All components can be either combined on a single chip implanted above the retina, or in two different chips with the simulating and electrode arrays placed above the retina [28] [29].

The advantage of the epiretinal implant is that the implant is off the retinal surface and exists in the vitreous cavity, which contains the vitreous fluid that helps dissipate the heat generated by the implanted electronics [29].

The disadvantages include the risk at the time of attaching the epiretinal implant to the retina. The implant experiences rotations of speeds of more than 400 degrees/second, due to the movement of the eyeball. There are many methods to attach the implant, including retinal tacks, but they are still under investigation. Another disadvantage is the non-focal widespread stimulation. The implant is placed on top of ganglion cell axons, which are in close proximity to each other, making stimulating particular axons difficult, and the task of encoding the image tricky since many ganglion cells will be stimulated at once. Also, it

is unknown how much power can be transmitted wirelessly to the implant without raising the temperature of the eye and thus, damaging it [29].

2.2.3 Optic Nerve Prosthetics

Optic nerve prosthesis is useful for patients who have a retina that is entirely degraded. The optic nerve is a closely packed bundle of ganglion cell axons. This bundle transports the electrical signal from the retina to the visual cortex for interpretation. A typical optic nerve prosthesis is shown in Figure 2.5 [29].

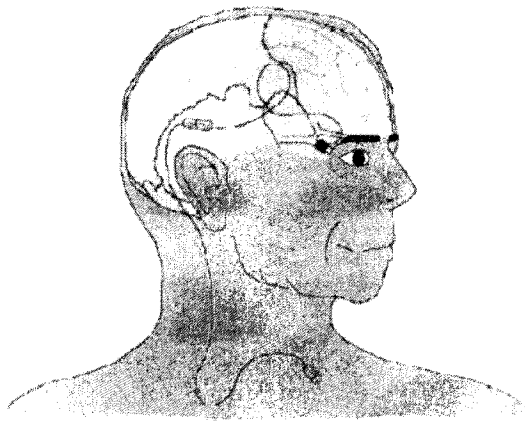


Figure 2.5: Optic nerve prosthesis [29]

The optic nerve prosthesis is called the Microsystem-based Visual Prosthesis (MiViP) and was developed at the University of Louvain in Brussels. This system consists of a spiral cuff electrode placed around the optic nerve. The electrode is connected to a thin wire that runs through a hole made in the patient's skull to a simulator implanted in a depression in the skull. The stimulator chip receives signals from an external camera and translates them into pulses that stimulate the optic nerve. Power is supplied externally [28].

One of the disadvantages of this approach is the risk involved in the surgical implantation of the device. This surgery can have harmful side effect such as infection, disease and disturbance of blood. Another disadvantage is the high density of axons in the optic nerve. As with the epiretinal prosthesis, it is difficult to provide focal simulation and detailed images [29].

2.2.4 Visual Cortical Implants

In visual cortical implants (VCI), only the visual cortex has to be intact. Unlike the other implants, the retina and optic nerve can be degraded or damaged. Patients born blind cannot use the implant, since the visual cortex would have been rearranged for other purposes. Yet, this implant is the most versatile one and can be used for many more blind patients [26].

A typical visual cortical implant is shown in Figure 2.6. As with the other implants, vision starts with a camera or an imaging device. The captured image is then processed to extract important features, such as edges [31].

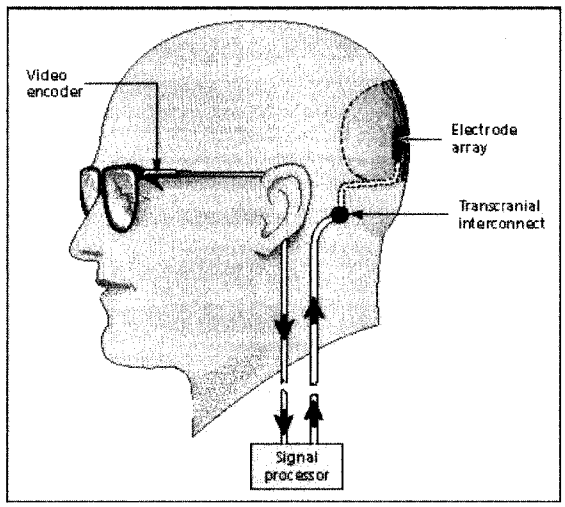


Figure 2.6: A typical visual cortical implant, showing the placement of the electrodes [31]

Once the important features are extracted, phosphene mapping takes place, by which an experiment is carried out on the patient. Phosphenes are evoked and the patient is asked to describe their location, and this draws up each patients phosphene map. Phosphenes are topographically induced visual sensations or dots of light. Due to the complex folded nature of the visual cortex, points on the visual field do not correspond to the same positions in the visual cortex. Figure 2.7 demonstrates how the traditional approach of one to one retinotopic mapping no longer applies due to new discoveries.

In the traditional view, it was believed that the image seen on the retina evokes neural activity in the visual cortex in the same shape. Contemporary views prove that the neural

pattern of activity is not as simple as a one-to-one correspondence, but rather a more sophisticated form. Thus, phosphene mapping has to take place before evoking sight. Experiments on the patient have to be carried out in order to draw their visual cortical map [31].

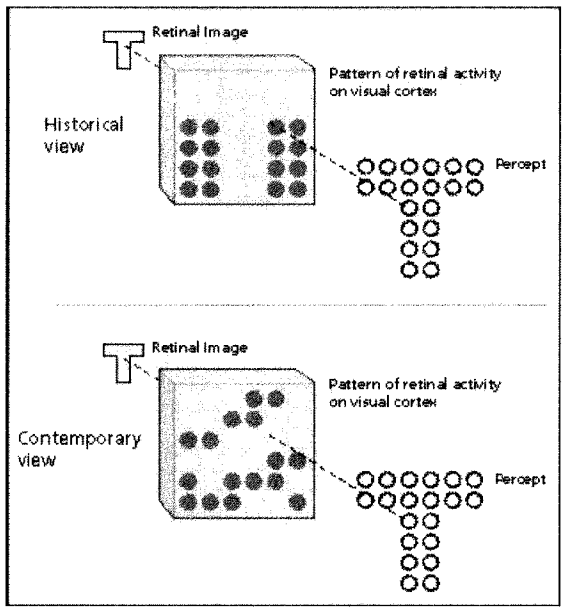


Figure 2.7: Retinotopic mapping: historical view and current view indicating random mapping [31]

To evoke sight, the resulting image is sent in the form of electrical signals to an electrode array, implanted at the visual cortex. The electrodes are composed of an array of needle-like simulators. Each electrode can both record from and stimulate the cortex. Needle electrodes are more common than dot electrodes due to the lower threshold voltage they use and the fact that neighbouring electrodes can be placed closer together without interference [26].

The brain has an amazing ability to reconfigure itself, that is why the electrodes should be designed to move tissue out of its way, as it is placed, and not cut through it. Since the brain moves independently of the skull, any wires connecting the electrodes should be given some allowance for movement [32]. Over time, the patient will be able to recognize more and more visual sensations. The adaptability of the brain should allow for better correlation between the physical world and the viewed phosphenes [31].

The major advantage of VCI is its versatility. It can be used on many blind patients, as long as their visual cortex is intact. Although the implant has been successfully tested on several patients, the surgical procedure still carries a higher risk than that of subretinal implants [26].

2.2.5 Parameters for Electrical Stimulation

When designing electrodes for implants, safety parameters have to carefully considered, many aspects are well thought out before the implant is used. This section covers parameters to be considered for electrical stimulation along the visual pathways, applicable to all prosthetics, which are influenced by the following [26]:

- Target cells, its anatomy, electrical properties and the portion of the cell being stimulated, affect the threshold voltage needed for stimulation .For example, the axon of a cell will have a different threshold than its soma.
- Stimulating electrode distance from the cell. Using larger electrodes or placing them closer to the cell helps reduce the charge density needed to keep it within safe limits for long term stimulation. Keeping in mind large electrodes lose resolution.
- For retinal stimulation, a pulse duration of 0.5ms is needed.
- As the pulse duration decreases, threshold current increases, and vice versa. Care should be taken not to reach the minimum value of threshold current, called the rheobase. At this value an action potential cannot be elicited regardless of the duration of the pulse.
- The repetition rate of electrical stimulus, for optic nerve implants it is 160 Hz, for visual cortical implants, it is 150-200 Hz.
- Several different polarities can activate neurons, such as cathodic, anodic or biphasic pulses. Each polarity has a different threshold value.
- Sinusoidal and pulsatile or square waveforms have been used frequently, but there are many variations being explored.

Neural response can improve by careful consideration of pulse parameters for the prosthesis in question [26].

Damage by Electrical Current

Several aspects have to be considered to minimize damage to the neural cells on any of the implants described above.The following point summarize the conditions under which damage can be avoided or caused [26]:

- DC current, or simple monophasic current, cause irreversible Faradic processes. Faradic processes involve a transfer of electrons across the electrode-tissue interface and oxidation/reduction of chemical. A biphasic current, containing two waveforms of equal charge but opposite polarity and contains no DC component, making it safer to use.
- Chemical reversibility, where a pulse of opposite polarity will chemically reverse all process occurring at the electrode. This ensures minimal damage to tissue and electrode.
- Damage is dependent on current amplitude, pulse repitition rate, charge density and charge per phase. Charge density is the charge per phase divided by the area of the electrode that is electrochemically active. Charge per phase is the integral of the stimulus current over one phase of a cycle of pulse duration. These two merits place a limit on how small the electrode can be made, without damaging stimulated cells.
- Heat damage, direct retinal stimulation cannot take more than 50 mW of power for 1.4 mm^2 area, where as 500 mW in the middle of the Vitreous can be kept up for 2 hours without damage.

In spite of the limitations stated above, electrical stimulation of neural cells is safe and stable, as long as the safety limitations of the implant are respected [26].

2.2.6 Conclusion

There are several prosthetics available for the blind. Of all the implants, visual cortical implants can be applied to most blind patients. VCI provides patients with phosphene vision, the surgery is simple and has been successful in all implant patients.

2.3 CMOS Imagers and CCD Imagers

This section covers the basics of Charge Coupled Devices (CCD) and CMOS imagers. It also provides a comparison between the two that explains the choice of imager for artificial retinas. But before examining the two imagers, a brief section presents some common terminology used in imaging devices.

2.3.1 Common Terminology for Imagers

The following is a list of important terms used in describing and comparing imaging devices and designs.

- Fill Factor (FF): the percentage of the pixel area that is photosensitive. The higher the fill factor, the more detailed the image. Lower fill factors can cause thermal noise due to the extra circuitry. A low fill factor also produces grainy or low resolution images [33].
- Dynamic Range (DR): quantifies the sensors' ability to image a scene in a range of illuminations. The lower the dynamic range, the more an image will miss in dark or bright illuminations. The DR of the human eye is 90 dB. Dynamic range is calculated as the ratio of the largest non-saturating photocurrent in a pixel $i_{\max}$, to its smallest detectable current $i_{\min}$, as shown in Equation (2.3.1) [33].

$$\mathrm{DR} = 20\log_{10}\frac{i_{\max}}{i_{\min}} \tag{2.3.1}$$

- Fixed Pattern Noise (FPN): noise due to interconnect and device mismatch and can affect the quality of the image [33].
- Dark Current: the leakage current of the photodiode when no illumination is applied. At low illuminations, dark current introduces shot noise and can affect image quality. Dark current is measured when the imaging device is fabricated [33].
- Antiblooming: sensor's ability to drain any locally overexposed pixel without affecting the rest of the image. It is also measured after fabrication [34].

2.3.2 CCD Imagers

Figure 2.8 shows a typical CCD imager. The CCD's image sensor is composed of photodiodes only. This gives the sensor a fill factor of 100%. Once fully exposed, the charge is read sequentially, buffered and converted externally. An external printed circuit board controls the sensor and processes the image. Most of the CCD's functions take place on the circuit board and can be easily changed as the design changes [34].

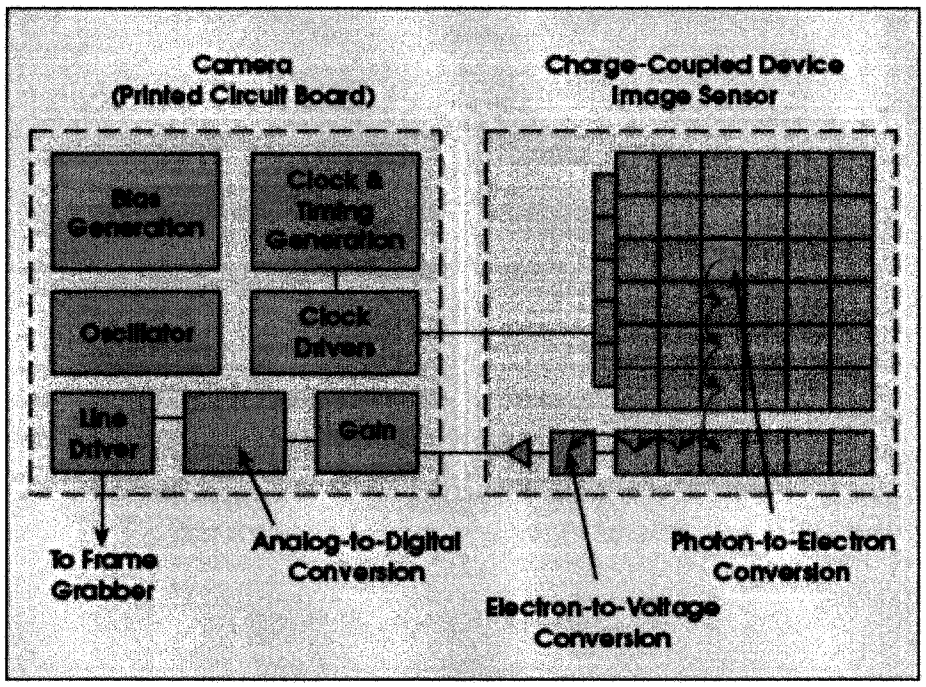


Figure 2.8: A typical CCD imager [34]

CCD sensors have many advantages and disadvantages when compared to CMOS imagers, and will be discussed later in this chapter. Their dynamic range spans 60-70 dB.

2.3.3 CMOS Imagers

CMOS imagers are increasingly competing with CCDs in many aspects. A typical CMOS sensor is shown in Figure 2.9. Integrated into the circuit is the entire function of the imaging device. Along side the photodiodes are charge converters, and in some cases, signal processing circuits. CMOS imagers are typically more reliable but less flexible than

CCDs. Their fill factor and dynamic range are summarized in Table 2.1.

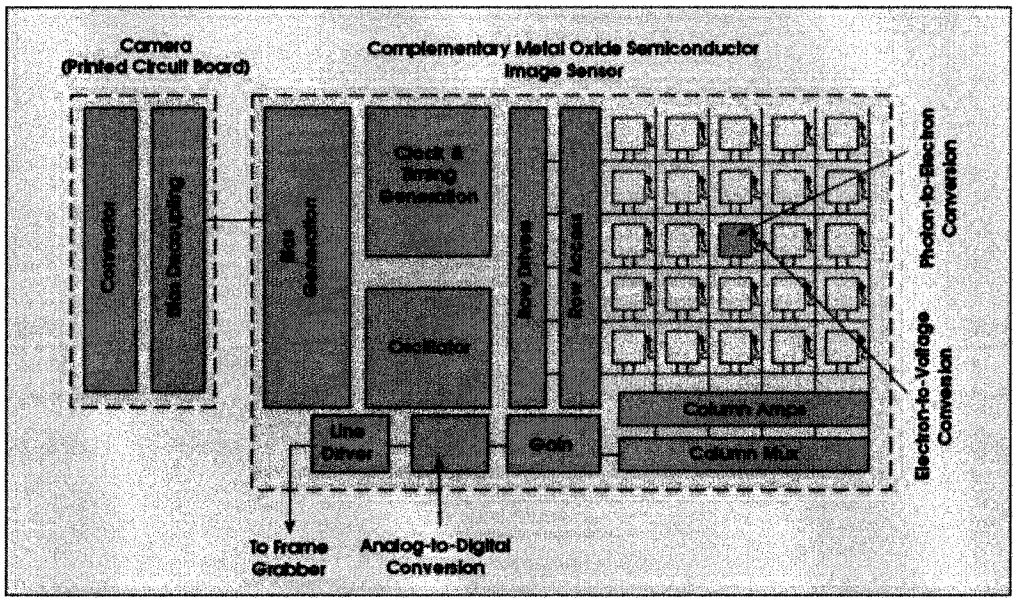


Figure 2.9: A typical CMOS imager [34]

Table 2.1: Fill factor and dynamic range for CCD and CMOS imagers

Type	FF	DR
PPS	$\leq 60\%$	40-60dB
APS	$\leq 50\%$	40-60dB
DPS, no PE	$\leq 25\%$	40-60dB
DPS, with PE	$\leq 10\%$	40-60dB
CCD	$\leq 100\%$	60-70dB

There are several kinds of CMOS imagers. Passive pixel sensors (PPS') were the first CMOS imagers. A typical pixel consists of a photodiode and a read out circuitry. This type of pixel was very slow and vulnerable to noise. The PPS is seldom used, and its problems are easily fixed in other CMOS imagers, such as APS and DPS, but at a cost of extra transistors [33].

Active Pixel Sensors (APS) are pixels that include reset, source follower and a row select transistors. The sensor data is read one row at a time and the pixels are reset after each read. The APS solves the problems of PPS and allows for automatic antiblooming and improved DR [33].

The most recent type of CMOS sensors are the Digital Pixel Sensors (DPS). These sensors contain a photodiode, reset circuity and on pixel analog to digital conversion. Some DPSs also include processing elements, which add more versatility to the chip. The main advantage of DPS over the analog sensors, APS and PPS, is the ability to better scale with CMOS technology, because of eliminated FPN and column readout noise. It also allows for massively parallel ADC and high-speed readout. The drawback to DPS is its low fill factor due to increased number of transistors [33].

DPSs are sometimes called "artificial retinas", mostly due to the fact that they process the image at the pixel level, just as a biological retina does. Some analog CMOS sensors also include analog processing elements. For example, some carry out multiplication on analog signals. Such sensor can also be labelled "artificial retinas". Thus, any sensor incorporating pixel-level signal processing can be said to be an artificial retina [33].

2.3.4 Comparison of the two imaging technologies

Each technology has its advantages and disadvantages, which makes each sensor suited to a different task, as shown in Figure 2.10 [35].

The advantages of CCD sensors are as follows [34]:

- Higher fill factor, thus allowing for high quality images
- More flexibility, thus no need to redesign the imager
- Better Dynamic range

Disadvantages of CCDs [34]:

- No windowing ability
- Need extra anti-blooming capabilities
- Need a specialized fabrication line
- Less reliable in rugged conditions
- Consumes more power in most cases
- Slower than CMOS

Advantages of CMOS imagers [34]:

- Windowing Ability
- Automatic Anti-blooming capability
- Uses regular CMOS fabrication lines

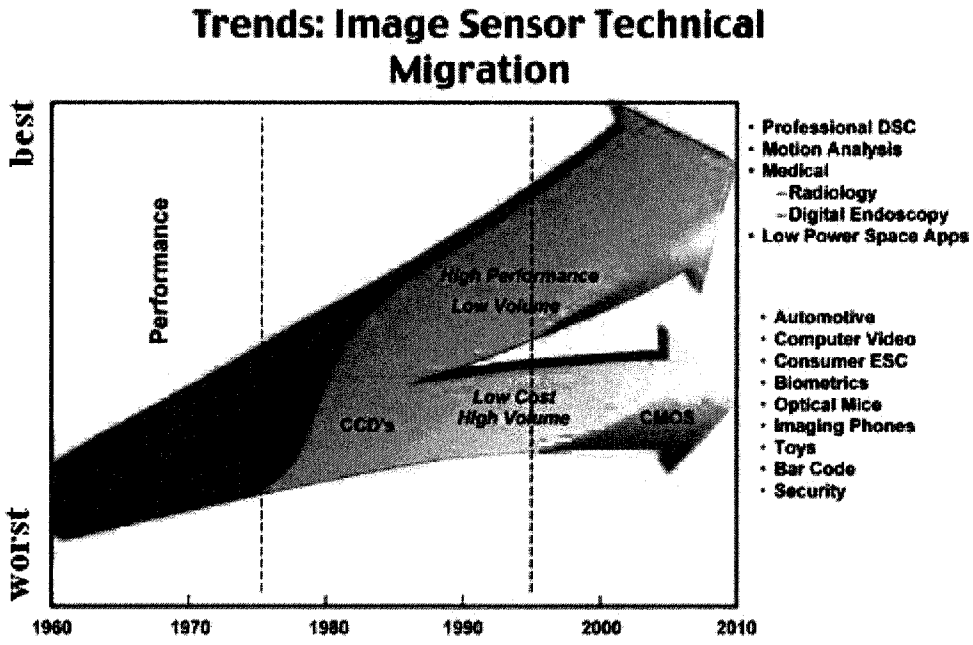


Figure 2.10: Future trends in CMOS and CCD imagers [35]

- More reliable in rugged conditions
- Consumes less power in most cases
- Faster than CCDs, allowing for real time imaging
- DPSs eliminate FPN

Disadvantage of CMOS imagers [34]:

- Lower fill factor
- Less flexible, the imager at times needs redesigning
- Dynamic range is not as good as CCDs

2.3.5 Conclusion

CCD imagers are superior in their image quality and applications needing high quality images are well suited for CCDs. On the other hand, CMOS imagers provide a better alternative for many more applications. Biometrics are amongst those applications, along with realtime imaging.

From the discussion in this chapter, it can be seen that CMOS sensors are the most appropriate imagers aimed at VCI. Since the resolution of the imager is limited by the electrodes implanted, A lower than 100% fill factor is acceptable. CMOS imagers reliability make them ideal for everyday use, and in many weather conditions. VCIs also need realtime ability to give the patient a more realistic view of the physical world.

Chapter 3

Algorithms for Edge Detection

3.1 Introduction

The goal of edge detection is to mark point of sharp luminous intensity change in an image. These changes mark important events, changes and objects in the scene. An edge detected image is an image where only the structural information of an image is preserved. Recall that the retina carries out edge enhancement, thus, knowledge of edge detection algorithms is a must for designing an artificial retina.

Since the focus of this thesis is hardware implementation, this chapter will not go into details but rather covers the basics of edge detection. It then moves on to describe some of the most popular algorithms: the Canny, Sobel and Roberts Cross edge detectors. For comparison, mathematical morphology , a simpler yet effective way for edge detection, will be covered. A summary of the discussion will be given in the conclusion.

3.2 Basics of Edge Detection

Segmentation is the identification of similar regions within an image [36]. Image segmentation is computationally demanding and was used to form edge images. As an alternative, edge detection has provided inherent advantages. These advantages include a substantial reduction in the space needed to store the information and the amount of processing needed [37].

Edge detection has been evolving for over 30 years, with many methods being constantly

improved or added. This chapter covers derivative operator, template and global model-based edge detections. But first, a definition of an edge is needed to understand how to find one [37].

3.2.1 Definition of an Edge

Several types of edges exist, viewpoint dependent edges change as the viewpoint changes and reflect important geometries of the scene. Whereas viewpoint independent edges reflect the properties of the object being viewed such as shape, geometry and markings. Another classification of edges include the ideal step edge and the ramp or graded edge are discussed in more detail below.

Ideal Step Edge

The ideal edge is a change in grey level at a specific position. Figure 3.1 shows a typical step edge. The x-axis denotes the position of the pixel, where as the y-axis is the gray-level of the pixel [36].

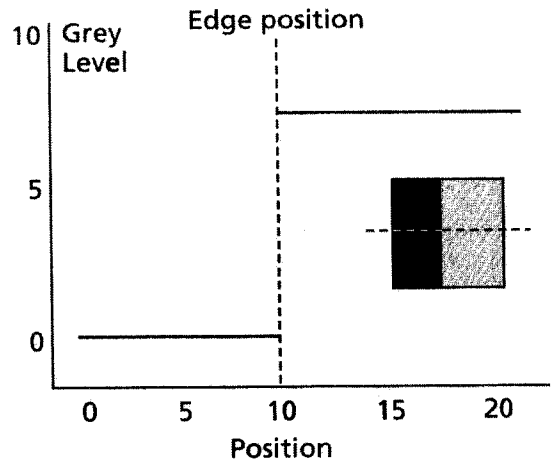


Figure 3.1: Typical ideal step edge [36]

The edge detector should find the edge at the position of grey level change, in this case, it occurs exactly at pixel position 10. The larger the change in grey level, the easier it is to detect the edge [36].

Ramp Edge

The ramp edge, or graded edge, is a result of digitization, and is more likely to be encountered in images. In this case, the change in pixel grey level occurs gradually and over several pixels. Figure 3.2 shows a typical ramp edge. The figure shows the same change in grey level, but this time, over 10 pixels. The centre of this change is pixel 10; thus, the edge, when detected, should be placed at pixel position 10 [36].

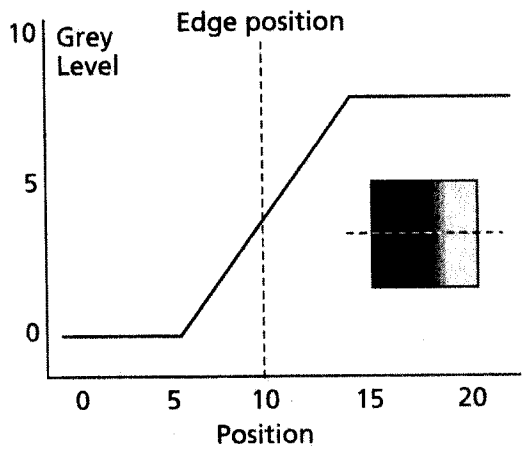


Figure 3.2: Typical graded edge, the change spans 10 pixels [36]

Many edge detection algorithms use a mask or structuring element on the image. The type of structuring element greatly affects the end result. By choosing the appropriate structuring element (SE) the desired effect can be achieved. The different algorithms and methods differ in the structuring element used. For example the Roberts Cross algorithm uses a two 2x2 masks, in the x and y directions to detect edges in an image. Others, such as the Canny use a gaussian based mask. There are also two types of operators that can be used, first order operators, such as the Roberts Cross, Prewitt, Sobel and Canny. Second order derivative operators include the Marr-Hildreth. [36] [37].

Some algorithms for edge detection involve calculating the intensity gradient magnitude, g, of the image. To find the gradient, the structuring element is convolved with the image. Equation (3.2.1) shows that by convolving an image with a specific SE or mask, the end result is the gradient of the image. It is the SE that determines how large the gradient

difference is in order to find an edge. Thus, where there is an edge, there is a large change in gradient magnitude, otherwise, the gradient magnitude is relatively unchanged [36] [37].

$$g = Image \otimes SE \tag{3.2.1}$$

In general, all algorithms use two SEs, one in the x-direction and another in the y-direction [36]. Thus, the above equation can be rewritten as

$$x - \text{gradient} : g_x = Image \otimes SE_x \tag{3.2.2}$$

$$y - \text{gradient} : g_y = Image \otimes SE_y \tag{3.2.3}$$

Once the two gradients are found, they are combined to find the edge gradient magnitude of the image. Equation (3.2.4) shows how the magnitude is computed vectorially. To save computational effort, Equation (3.2.5) behaves similarly can be used [37].

$$g = (g_x^2 + g_y^2)^{1/2} \tag{3.2.4}$$

$$g = |g_x| + |g_y| \tag{3.2.5}$$

The next sections discusses the different types of methods in edge detection. The discussion will then move on to describe two popular algorithms.

3.3 Derivative Operator-Based Edge Detection

Derivative operator-based edge detection uses the principle of rate of change. The rate of change in grey levels in an image is large near an edge, and small or constant otherwise [36].

To detect the edge, these types of algorithms take the partial derivative of the image in the x and y directions. The resulting x and y gradients are combined using Equation (3.2.4) or (3.2.5). This type of edge detection is poor and there are many better methods, such as the template-based and the global model-based edge detection [36].

3.4 Template-Based Edge detection

The structuring element in template-based edge detectors is a simple small discrete matrix. The method attempts to approximate the gradient of the pixel relative to the centre of the template. Two of the algorithms that employs template-based edge detection are the Roberts Cross and the Sobel algorithm, discussed next [36].

There are several design goals that Template-based structuring elements aim for. There is always a conflict with the goals, thus tradeoffs and compromises have to always be made. The design goals are the following [37]:

- The accuracy of edge magnitude estimation has to be optimized
- The accuracy of edge orientation estimation has to be optimized
- Suppress noise during edge detection
- optimize different structuring elements for different edge types

3.4.1 The Roberts Cross Algorithm

One of the earliest edge detection algorithms is the Roberts Cross. Its is based on computing the sum of the squares of the difference in the diagonally adjacent pixels. This is done by convolving the image with two 2x2 SEs in the x and y direction, given in (3.4.1) and (3.4.1).

$$R_x = \begin{matrix} 0 & 1 \\ -1 & 0 \end{matrix} \tag{3.4.1}$$

$$R_y = \begin{matrix} 1 & 0 \\ 0 & -1 \end{matrix} \tag{3.4.2}$$

Although the Roberts Cross algorithm is simple and fast to implement, it is susceptible to noise due to its small SE. Its results are not as impressive as Sobel, but it is able to detect edges and give adequate results.

3.4.2 The Sobel Algorithm

The templates used in the Sobel algorithm are shown in (3.4.3) and (3.4.4). It can be seen that less weight is given to the diagonal components of the mask [36].

$$S_x = \begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix} \tag{3.4.3}$$

$$S_y = \begin{matrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix} \tag{3.4.4}$$

The templates above are applied to every pixel in the image. The resulting x and y gradient magnitudes are computed using (3.2.4) or, most likely, (3.2.5). The result is the thresholded; all pixels will respond to the SE, but only those with the largest response correspond to an edge [36].

Thresholding refers to setting a gray-scale value as a limit, any pixel with a gray-scale value under this limit is set to 0, all values over are set to 1. Thus the image is turned into a binary image.

3.5 Global Model-based Edge detection

Global Model edge detectors use a more complex global structuring element for their algorithms. They make up the most complex but the most powerful method. Each algorithm uses a different type of SE, and different approaches to applying it. One of the most popular global model detectors is the Canny algorithm, which will be described next.

3.5.1 The Canny Algorithm

In 1986, John Canny, described an optimal edge detection algorithm by setting goals for edge detection. He defined three issues that an edge detection algorithm should achieve [36]:

- Error rate: should find all edges, and respond only to edges
- Localization: the distance between the detected edge and actual edge should be minimum
- Response: multiple edges should not be detected when a single edge exists

To achieve these goals set above, while maximizing the signal to noise ratio (SNR), Canny found that the SE is the first derivative of the Gaussian function, where σ is the

standard deviation of the gaussian function [36].

$$G(x) = e^{-\frac{x^2}{2\sigma^2}} \tag{3.5.1}$$

Since the image has both x and y components, the two dimensional gaussian, Equation (3.5.2), must be [36].

$$G(x,y) = \sigma^2 e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.5.2}$$

The Canny algorithm uses the gaussian derivatives in both the x and y directions. Thus, to find an edge image using the Canny algorithm, the following steps should be taken [36]:

1. Read in the image
2. Smooth the image using a 1D Gaussian mask G
3. Create 1D mask for the first derivative of the Gaussian G_x, in the x direction, and G_y, in the y direction
4. Create a second 1D mask for the first derivative of a second Gaussian G'_x, in the x direction, and G'_y, in the y direction with a similar σ
5. Convolve the image with the mask G in the x direction, as follows: $I_x = Image \otimes G_x$. Then convolve the image with the mask G in the y direction, as follows: $I_y = Image \otimes G_y$
6. Convolve the results above with the gaussian x and y masks. $I'_x = I_x \otimes G'_x$ and $I'_y = I_y \otimes G'_y$
7. Combine the I'_x and I'_y components according to equation 3.2.4 to make the image.
8. OR, in place of the second convolution, Carry out edge detection algorithm such as the Robert Cross or the Sobel
9. Any non-maxima pixels are suppressed to thin the edges. A pixel is suppressed only if its gradient magnitude is smaller than its neighbours

10. The resultant image is threshold using two different thresholds, τ_1 and τ_2, where $\tau_1 < \tau_2$. The two binary images obtained are T_1 and T_2, where T_2 is has less noise and false edges, but large gaps between the edge segments.

11. Trace the edge segments in T_2 the their end, then search T_1 to find edge segments to bridge the edge gaps in T_2.

Canny introduced the concept of non-maximum suppression. In non-maximum suppression, edge points are points where the gradient magnitude is a maximum in the gradient direction. This can be carried out by estimating the gradient direction using first order derivatives, then the result is rounded off to a multiple of 45 degrees. The last step is to compare the value of the gradient magnitude, at that point, in the direction estimated. If the point has the largest gradient magnitude, then the point is kept, otherwise it is eliminated [36].

The canny algorithm is dependent on σ, where by controlling σ controls the size of the Gaussian filter. The larger σ is, the larger the filter, this causes more blurring suitable for noisy images, but a less accurate localization of the edge. Decreasing σ minimizes bluring and produces finer edges.Thus σ should be adjusted according to the environment and to the noise levels in the image [36].

3.6 Mathematical Morphology

Morphology is the study of the form and structure of an image [36]. Mathematical morphology studies the mathematical properties of shapes and objects. By using simple equations images can be morphed and changed to achieve any desired result. The two most important morphological operations are erosion and dilation. Many other operations, such as close and open, can be derived from erosion and dilation; but will not be covered in this thesis since they are not needed for edge detection [37] [38].

Before describing these operations, the numbering scheme for a typical pixel and its neighbours should be introduced. In Table 3.1, the centre pixel, P0, is the pixel of interest, or the current pixel; all other pixels around it are its neighbours, and are numbered starting from the right, moving counter-clockwise [37].

Table 3.1: Standard layout and numbering of pixels

P4	P3	P2
P5	P0	P1
P6	P7	P8

There are three types of mathematical morphology. Binary morphology, which is carried out on binary images, where 1 is white and 0 is black. Gray-scale morphology, carried out on gray-scale images, where 255 is white and 0 is black, where the values in-between are different shades of grey. Finally, there is colour morphology on colour images, but this is more complex and will not be covered in this thesis. The next sections will cover only erosion and dilation for binary and gray-scale morphology [36] [37].

3.6.1 Binary Mathematical Morphology

Dilation

Dilation expands the binary image into the background and removes any cracks that are less than three pixels wide. In binary images, dilation is simple. Equation (3.6.1) show how to calculate the dilation, where P are the pixels of the original image, and D represents the dilated image [37].

$$\begin{aligned} &Dilation : [Sigma = P1 + P2 + P3 + P4 + P5 + P6 + P7 + P8; \\ &If(Sigma > 0)D0 = 1; \\ &elseD0 = P0] \end{aligned} \tag{3.6.1}$$

In more simple terms, with a 3x3 SE of all ones, binary dilation is the OR of all the neighbouring pixels with the current pixel [13] [38].

$$Dilation : [D0 = P0||P1||P2||P3||P4||P5||P6||P7||P8] \tag{3.6.2}$$

Erosion

Erosion shrinks the binary image and removes any thin objects that are less than three pixels wide. In binary images, erosion is simple. Equation (3.6.3) shows how to calculate the erosion, where P is the pixels of the original image, and E is the eroded image [37].

$$Erosion : [Sigma = P1 + P2 + P3 + P4 + P5 + P6 + P7 + P8;$$
$$If(Sigma < 8)E0 = 0; \tag{3.6.3}$$
$$elseE0 = P0]$$

In more simple terms, with a 3x3 SE of all ones, binary erosion is the AND of all the neighbouring pixels with the current pixel [13] [38].

$$Erosion : [E0 = P0\&P1\&P2\&P3\&P4\&P5\&P6\&P7\&P8] \tag{3.6.4}$$

3.6.2 Gray-scale Mathematical Morphology

Gray-scale morphology is slightly more complex than binary morphology, but it is still quite simple. The SE for gray-scale images can contain any gray-scale value. For simplicity, it is easier to use a flat SE that is, only 1 and 0 are used.

Dilation

Where the image is denoted with f, and the SE with h, dilation is usually written as $f \oplus_g h$. In gray-scale, dilation is carried out by setting an intensity maximum on the pixel using the SE, as shown in Equation (3.6.5). In this Equation, the coordinates of the current pixel are (r, c), (i, j) are all in the domain of h, and $(r - i, c - j)$ is in the domain of f [39].

$$f \oplus_g h = \max_{i,j}(f(r - i, c - j) + h(i, j)) \tag{3.6.5}$$

If the structuring element is binary, or flat, then Equation (3.6.5) can be simplified to Equation (3.6.6)

$$f \oplus_g h = \max_{i,j}(f(r - i, c - j)) \tag{3.6.6}$$

Erosion

Similarly, erosion can be written as $f \ominus_g h$. In gray-scale, it is carried out using an intensity minimum as shown in Equation (3.6.7) [39].

$$f \ominus_g h = \min_{i,j}(f(r + i, c + j) - h(i, j)) \tag{3.6.7}$$

The minimum here is taken where the coordinates of the current pixel are (r, c), (i, j) are all in the domain of h, and $(r + i, c + j)$ is in the domain of f. Once again, if the SE is flat, then Equation (3.6.8) can be applied.

$$f \ominus_g h = \min_{i,j}(f(r + i, c + j)) \tag{3.6.8}$$

As an example to demonstrate Equations (3.6.6) and (3.6.8), Figure 3.3 a) shows a graded edge. In this image, 0 is black and 60 is white. In gray-scale erosion, the image, which is white, is eroded, thus, the image becomes darker, that is, more 0 values, as seen in Figure 3.3 b) . On the other hand, gray-scale dilation expands the white parts, making the image brighter as in Figure 3.3 c).

60	60	60	55	55	0
60	60	55	55	0	0
60	55	55	0	0	0
55	55	0	0	0	0
55	0	0	0	0	0
0	0	0	0	0	0

Test Image

(a) Test image

60	60	60	60	55	55
60	60	60	55	55	0
60	60	55	55	0	0
60	55	55	0	0	0
55	55	0	0	0	0
55	0	0	0	0	0

Dilation

(b) Dilated image

60	60	55	55	0	0
60	55	55	0	0	0
55	55	0	0	0	0
55	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Erosion

(c) Eroded image

Figure 3.3: An example of gray-scale mathematical morphology

3.6.3 Edge Detection Using Mathematical Morphology

To find the edge using mathematical morphology either erosion, dilation, or both can be used [37].

Since erosion is a shrunken version of the image. The edge is found by subtracting the erosion from the image,Equation 3.6.9, this leaves the edge pixels. [37]:

$$E = P - (P \ominus B) \tag{3.6.9}$$

Dilation expands the binary into the background. Subtracting the image from the dilation will also produce an edge image, Equation 3.6.10 [37]:

$$E = ((P \oplus B) - P) \tag{3.6.10}$$

Equation 3.6.11 shows how both erosion and dilation can be used for edge detection [37]:

$$E = (P \oplus B) - (P \ominus B) \tag{3.6.11}$$

The difference between the three methods of edge detection would be seen in dilation. The edge produced from Equation 3.6.10 is offset from its original position. Using both erosion and dilation as in Equation 3.6.11 would produce a more noise resistant edge image.

3.7 Conclusion

There are many algorithms for finding edges in an image, only a small portion of which were discussed in this chapter. The first method explored in this chapter is the use of classical algorithms, such as the Canny and Sobel. These algorithms are powerful and yield good results. Their main drawback lies in their computationally intensive nature.

The second method of edge detection explored is through mathematical morphology. The images that they yield are more diffused, but they detect all the edges. Compared to classical algorithms, mathematical morphology is more simple and easy to implement. They also have the ability to reduce Salt and Pepper noise, discussed in Chapter 4, common in DPS due to digital conversion.

Thus, in the context of a portable imager for visual cortical implants, the more attractive option is mathematical morphology, since it allows better integration of image processing functions on-pixel.

Chapter 4

A New Method of Gray-scale Erosion For On-Pixel Implementation in Imaging Devices

4.1 Introduction

This chapter introduces a new gray-scale erosion method. This method simplifies gray-scale morphology for implementation on-pixel. Although the proposed erosion can also be expanded into dilation, erosion was chosen since the detected edge is more accurate as seen in chapter 3. The pixel implements erosion as needed in a more noise robust algorithm for edge detection using gray-scale mathematical morphology.

4.2 Proposed Gray-scale Bitwise Decomposition

4.2.1 Gray-scale Threshold Decomposition

Before the proposed method of gray-scale erosion is introduced, a brief background into threshold decomposition will be given.

The objective of gray-scale threshold decomposition is to take a gray-scale image, then derive several binary images from it. This allows for the application of binary mathematical morphology. The gray-scale image is then recomposed using threshold superposition.

Decomposing the image produces M binary images, where M is the highest decimal value any pixel in the image may have. For example, an image with 4 bits, has a maximum value of 15, starting from 0. In this case M would equal 15. If an image has 4 bits, but

the maximum value in it is 10, then M would be 10. To decompose an image, Equation (4.2.1) is used, where $1 \leq m \leq M$, $f(x, y)$ is the original image, and $f_m(x, y)$ are the binary images. The binary image generated at $m = 0$ is composed entirely of 1's, which will not affect the result, that is why m starts at 1.

$$f_m(x, y) = \begin{cases} 1, & \text{if } f(x, y) \geq m, \\ 0 & \text{if } f(x, y) < m \end{cases} \tag{4.2.1}$$

This means that there would be M binary images generated by Equation (4.2.1). Similarly, if the structuring element used is gray-scale, it can be decomposed. In this application, a binary or flat structuring element is used, which simplifies the process.

Once these binary images are available, to carry out binary erosion, the method of threshold decomposition from gray-scale erosion to binary erosion is adopted from [41]. The mathematical proof of threshold decomposition is explained in details and derived. The end result is displayed in Equation (50) of [41], where it is proved that gray-scale morphology can be implemented as a series of binary morphological operations using the binary images generated from threshold decomposition. Since a binary SE is used, Equation (50) of [41] is reduced to Equation 4.2.2, where h is the SE.

$$f \ominus_g h = \sum_{m=1}^{M} f_m \ominus_b h \tag{4.2.2}$$

Similarly for dilation, as proven in [41], after decomposing the gray-scale image, binary dilation can be carried out according to Equation 4.2.3 for a binary SE.

$$f \oplus_g h = \sum_{m=1}^{M} f_m \oplus_b h \tag{4.2.3}$$

To carry out binary erosion on gray-scale images, with a binary SE, the following steps have to be taken:

1. Carry out threshold decomposition according to Equation (4.2.1)
2. Carry out binary morphology, erosion or dilation
3. Sum the resultant binary images

To illustrate this, consider the image in (4.2.4) and the structuring element in (4.2.5). This image has 2 bits, that is 4 possible values starting from 0, giving M a value of 3, where 0 is a black pixel, 3 a white pixel and 1 and 2 gray-levels in-between.

$$\text{Image} = \begin{matrix} 0 & 1 & 1 & 3 \\ 0 & 3 & 3 & 2 \\ 2 & 1 & 1 & 2 \\ 2 & 3 & 3 & 3 \end{matrix} \tag{4.2.4}$$

$$\text{SE} = \begin{matrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{matrix} \tag{4.2.5}$$

This image can also be visualized as that in Figure 4.1. The array has been replaced with gray-level pixels.

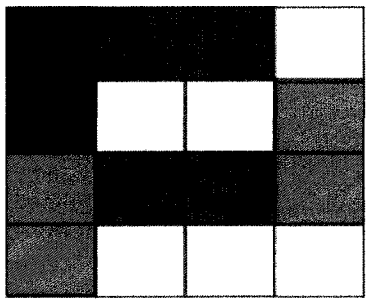


Figure 4.1: Visualization of the example given in (4.2.4)

Starting with $m = 1$, using Equation (4.2.1), a binary image is generated, given in (4.2.6).

$$\text{m1} = \begin{matrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{matrix} \tag{4.2.6}$$

Similarly, for $m = 2$ and $m = 3$, (4.2.7) and (4.2.8) are generated respectively.

$$\text{m2} = \begin{matrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{matrix} \tag{4.2.7}$$

$$\text{m3} = \begin{matrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{matrix} \tag{4.2.8}$$

Following the outlined steps, the binary SE in (4.2.5) is applied using binary erosion. The binary erosion applied here being ANDing the pixels at which the SE is 1. The results produced for $m = 1$, $m = 2$ and $m = 3$ are (4.2.9),(4.2.10) and (4.2.11) respectively.

$$m1 \ominus_b h = \begin{matrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{matrix} \tag{4.2.9}$$

$$m2 \ominus_b h = \begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{matrix} \tag{4.2.10}$$

$$m3 \ominus_b h = \begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix} \tag{4.2.11}$$

Adding up the values in each binary array produces the final result, in (4.2.12). This result equals that produced using the usual gray-scale erosion equations. A visualization of the image is shown in Figure 4.2. Since all-white space is less than or equal to three pixels wide, it is eliminated from the image.

$$\text{Eroded Image} = \begin{matrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 2 \\ 0 & 1 & 1 & 1 \\ 2 & 1 & 1 & 2 \end{matrix} \tag{4.2.12}$$

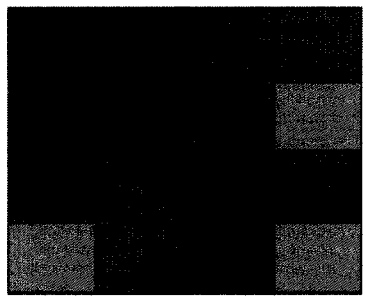


Figure 4.2: Visualization of the eroded image of (4.2.12) using threshold decomposition

4.2.2 Bitwise Gray-scale Decomposition for On-Pixel Implementation

The aim of the proposed Gray-scale decomposition is to provide a more feasible approach for on-pixel implementation of gray-scale erosion. To achieve this goal, the following points are considered:

- Reduce the number of binary images generated by the decomposition
- The result should be easily implemented in hardware, on-pixel
- Aim of the application is edge detection in visual cortical implants
- The aim being visual prosthetics, exact detail in the image is not necessary, but the overall shape should be recognizable

Gray-scale threshold decomposition produces M binary images. This means that for an 8-bit image, 255 levels have to be produced, resulting in an impractical hardware implementation for inclusion in pixels. The bitwise gray-scale decomposition provides for a better implementation, since only a multiplexer is needed to generate the binary images. Given that the number of bits representing each pixel is denoted by N, then the threshold decomposition produces $2^N - 1$ binary images. The proposed bitwise decomposition

produces N binary images.

To produce the binary images, each image is produced by a bit of each pixel in the original gray-scale image. This is achieved by following Equation (4.2.13), where $0 \leq n \leq N-1$.

$$\begin{aligned} f_0(x,y) &= f_{\text{bit0}}(x,y) \\ f_1(x,y) &= f_{\text{bit1}}(x,y) \\ \vdots \qquad &\qquad \vdots \\ f_n(x,y) &= f_{\text{bit n}}(x,y) \end{aligned} \tag{4.2.13}$$

The results of Equation (4.2.13) are N binary images. Now that binary images are available, binary erosion, or dilation can take place. The rules of binary erosion and dilation can also be applied to the process. Since this thesis is concerned with erosion, only the case with erosion will be presented, but this method can also be expanded to dilation, as shown later on in this chapter.

Equation (4.2.14) demonstrates how binary erosion can be carried out on each binary image produced from bitwise decomposition. The eroded binary image is denoted by E_n and h is the SE.

$$\begin{aligned} E_0 &= f_0 \ominus_b h \\ E_1 &= f_1 \ominus_b h \\ \vdots \qquad &\qquad \vdots \\ E_n &= f_n \ominus_b h \end{aligned} \tag{4.2.14}$$

The results of Equation (4.2.14) are N binary eroded images. Since the end result should be a gray-scale image, it has to be reconstructed from the binary images. Keeping in mind that the structuring element is flat, since f_0 is f_{bit0}, then it implies that $f_0 \ominus_b h$ is $(f_0 \ominus_b h)_{\text{bit0}}$ and so on for n from 0 to N. This means that E_0 to E_n represent bits 0 to N of the eroded gray-scale image. The result is a reconstructed eroded gray-scale image.

Demonstrating this, the method of bitwise decomposition is applied to the image in (4.2.4) and the structuring element in (4.2.5). Two binary images are produced in (4.2.15) and (4.2.16).

$$f_1 = \begin{matrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{matrix} \tag{4.2.15}$$

$$f_1 = \begin{matrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{matrix} \tag{4.2.16}$$

Applying binary erosion results in (4.2.17) and (4.2.18).

$$f_0 \ominus h = \begin{matrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{matrix} \tag{4.2.17}$$

$$f_1 \ominus h = \begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{matrix} \tag{4.2.18}$$

Reconstruction of the image is produced bit by bit as shown in (4.2.19) and also visualized in Figure 4.3.

$$\text{Result of using bit-wise AND for erosion} = \begin{matrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 2 \end{matrix} \tag{4.2.19}$$

It can be seen that the image is not exactly the same as the one produced by threshold decomposition. But, the result is, an eroded image of the original. In this case, more erosion is carried out, resulting in an even darker image than that of threshold decomposition. The most important result is that the white space that is less than 3 pixels wide is removed.

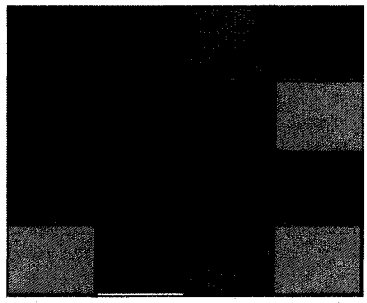


Figure 4.3: Visualization of the eroded image of (4.2.12) using bitwise decomposition

Also, since the resultant image is converted to binary, both eroded images would result in the same image.

A small range for the gray-scale levels is not ideal for bitwise decomposition. Since the application uses 8-bits to represent each pixel, this provides for a better use of the method. To demonstrate, Figure 4.4 shows an edge, where both dilation and erosion are carried out using bitwise decomposition, where 0 is black, 60 is white and all other numbers are gray-levels in-between. The result is a properly dilated and eroded image. When the image is converted to binary, all pixel values below 15 are turned to 0 (black), those above 15 to 1 (white). The value, 15, can be set to any number. Converting gray-scale to binary is called thresholding, not to be confused with threshold decomposition.

4.2.3 Advantages of Gray-scale Erosion Using Bitwise Decomposition

When it comes to hardware implementation, bitwise decomposition has several advantages, including the following:

1. Reduces the number of binary images produced from M to N, where $M = 2^N - 1$ and N is the number of bits representing each pixel in the image
2. Eliminates the need for decomposition circuitry, since the binary images are generated by a bitwise read of the image
3. Allows for binary erosion
4. Can be implemented on pixel, while maintaining a good fill factor

60	60	60	55	55	0
60	60	55	55	0	0
60	55	55	0	0	0
55	55	0	0	0	0
55	0	0	0	0	0
0	0	0	0	0	0

Test Image

(a) Test image

60	60	63	63	63	63
60	63	63	63	63	0
63	63	63	63	0	0
63	63	63	0	0	0
63	63	0	0	0	0
63	0	0	0	0	0

Bit-wise OR
Dilation

(b) Dilation with bitwise decomposition

60	60	52	52	0	0
60	52	52	0	0	0
52	52	0	0	0	0
52	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Bit-wise AND
Erosion

(c) Erosion with bitwise decomposition

Figure 4.4: An example demonstrating dilation and erosion using bitwise decomposition

5. Is well suited to edge detection for implementation in visual cortical implants

4.3 Improved Morphological Edge Detection Algorithm

Once the erosion is produced, edge detection using mathematical morphology can take place. There are several options that can be considered. The edge detection algorithms presented for mathematical morphology are simple, but they are not effective at eliminating noise, since they are sensitive to noise, since the operation can also dilate and/or erode the noise as well. Since robustness and reliability are important in the design of the artificial retina, a more noise immune algorithm must be used.

By using two structuring elements, the algorithms proposed in [39], are shown to reduce noise. This has been shown and proven in a series of tests mentioned in [39]. The paper provides two algorithms, one for dilation, the other for erosion. The two structuring

elements are shown in Equation 4.3.1 and 4.3.2, where a1 to a4 are gray-scale values.

$$\text{SE1} = \begin{matrix} 0 & a4 & 0 \\ a1 & 0 & a2 \\ 0 & a3 & 0 \end{matrix} \tag{4.3.1}$$

$$\text{SE2} = \begin{matrix} a1 & 0 & a4 \\ 0 & 0 & 0 \\ a3 & 0 & a2 \end{matrix} \tag{4.3.2}$$

The equation used for dilation is stated in Equation (4.3.3) and that for erosion in Equation (4.3.4), where A is the image, r and c are the coordinates of the current pixel [39].

Dilation :

$$G_d(r,c) = \min\left\{dilation_{SE1}(r,c) - A(r,c), dilation_{SE2}(r,c) - A(r,c), G_d''(r,c)\right\}$$

where (4.3.3)

$$G_d'' = \max\left\{ \begin{matrix} |(dilation_{a1}(r,c) - A(r,c)) - (dilation_{a2}(r,c) - A(r,c))|, \\ |(dilation_{a3}(r,c) - A(r,c)) - (dilation_{a4}(r,c) - A(r,c))| \end{matrix} \right\}$$

Erosion :

$$G_e(r,c) = \min\left\{A(r,c) - erosion_{SE1}(r,c), A(r,c) - erosion_{SE2}(r,c), G_e''(r,c)\right\}$$

where (4.3.4)

$$G_e'' = \max\left\{ \begin{matrix} |(A(r,c) - erosion_{a1}(r,c)) - (A(r,c) - erosion_{a2}(r,c))|, \\ |(A(r,c) - erosion_{a3}(r,c)) - (A(r,c) - erosion_{a4}(r,c))| \end{matrix} \right\}$$

To demonstrate Equations (4.3.3) and (4.3.4), Figure 4.5 depicts an edge in a test image. Both dilation and erosion edge detection are carried out on the image according to Equations (4.3.3) and (4.3.4). The structuring elements used are those in (4.3.5) and (4.3.6). Once again, the erosion edge image produces an edge where the original edge is located, whereas the dilation edge image produces an offset edge.

$$\text{SE1} = \begin{matrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{matrix} \tag{4.3.5}$$

$$\text{SE2} = \begin{matrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{matrix} \tag{4.3.6}$$

60	60	60	55	55	0
60	60	55	55	0	0
60	55	55	0	0	0
55	55	0	0	0	0
55	0	0	0	0	0
0	0	0	0	0	0

Test Image

(a) Test image

0	0	0	5	0	55
0	0	5	0	55	0
0	5	0	55	0	0
5	0	55	0	0	0
0	55	0	0	0	0
55	0	0	0	0	0

(b) Edge image using dilation

0	0	5	0	55	0
0	5	0	55	0	0
5	0	55	0	0	0
0	55	0	0	0	0
55	0	0	0	0	0
0	0	0	0	0	0

(c) Edge image using erosion

Figure 4.5: An example demonstrating edge detection using Equations (4.3.3) and (4.3.4)

4.4 Edge Detection Using Bitwise Decomposition

The edge detection algorithm used in this thesis is Equation (4.3.4). Unlike the Canny and Sobel algorithms, the algorithm does not carry out non-maximum suppression, a complex algorithm where pixels that are not a local maxima are removed. The result of non-maximum suppression is a cleaner image. This is done to save on computing power. If the implant has processing power to spare, it can be carried out.

The resultant edge image is that of Equation (4.3.4) output without manipulations. It will be seen below that, although no image cleaning up was done, the resultant image is comparable with the Canny and Sobel algorithms, but results in less computational effort.

Taking the same test image in Figure 4.5 a), bitwise decomposition into binary erosion and dilation is carried out and the results in Equation (4.3.3) and (4.3.4). The structuring elements used are (4.3.5) and (4.3.6). The result is shown in Figure 4.6.

0	0	3	3	8	63
0	3	3	8	63	0
3	3	8	63	0	0
3	8	63	0	0	0
8	63	0	0	0	0
63	0	0	0	0	0

(a) Edge image using dilation

0	0	8	3	52	0
0	8	3	52	0	0
8	3	52	0	0	0
3	52	0	0	0	0
52	0	0	0	0	0
0	0	0	0	0	0

(b) Edge image using erosion

Figure 4.6: An example demonstrating edge detection using bitwise decomposition

The only difference between the edge detected using bitwise decomposition and regular gray-scale erosion is a slight difference in gray-levels. The range of the image is not affected, the range refers to the values that represent the gray-scale pixels, and the edge is detected as expected. This approach would also remove any small insignificant pixels in the image.

4.4.1 Implementing Gray-scale Erosion Using AND Gates

Figure 4.7 shows an example of the implementation of the proposed erosion. The erosion is carried out bitwise, starting with the least significant bits. This example shows the SE having a 1 value on the top left and right corners. Thus, the hardware implementation is an AND gate connected to those pixels. The gates are fed the values one bit at a time. The output is put together one bit at a time to form the erosion result. This method works only for flat SE, where the only values are either 1 or 0. If a gray-scale SE is used, then the circuitry would be more complicated and would have to follow the equations outlined in [41]

By using the bitwise erosion shown above, edge detection is achieved by using Equation (4.3.4), where all the erosion operations mentioned in the equation are bitwise ANDs. The

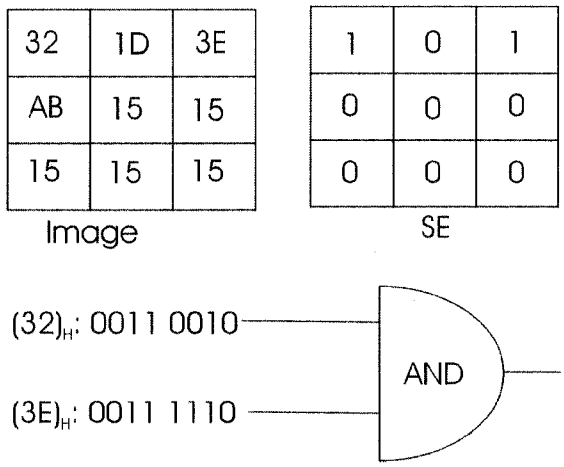


Figure 4.7: Implementation of gray-scale erosion using bitwise decomposition

algorithm uses two structuring elements: SE1 in (4.3.5) deals with the top, bottom, left and right neighbours, where as SE2 in (4.3.6) the diagonals.

The design uses two erosion results, each from one of the structuring elements. These results can be placed in Equation (4.3.4) to form the edge image.

To test out the algorithm, the Matlab code provided in Appendix B is written. Appendix B also provides the commands to run and compare the code as will be shown in the following sections. Several images were chosen, loaded into Matlab and the algorithm was executed. The resultant image was compared with those using the Canny and Sobel algorithms.

4.4.2 Testing Bitwise Gray-scale Decomposition in Edge Detection

This section will display several images, in gray-scale, along with their edge images, using the proposed method, mathematical morphology in Equation (4.3.4), the Canny and Sobel algorithms. It is tricky to compare the results of edge detection algorithms buy sight. But typically they are judged on how well the edge detector markings match objects in the scene or object boundaries. An edge detector is also compared based on its application. Since the application for this edge detection are visual prosthetics, then it can be compared on the level of details the algorithm produces. The amount of detail should be reasonable,

and the end result should be an edge representation of the scene being imaged. Following each set of images, a brief discussion will follow.

The first image is of a nature scene, namely a fern, by Jeremy Thomason [42]. Figure 4.8 displays four images, the original, Canny, Sobel and Proposed Algorithm. The edge image by the Canny algorithm shows all the edges. The connected lines create very confusing details on the fern. The image loses its texture, and looks less like a fern. Sobel edge detection, due to its simple SE, misses many of the edges. The image has too little detail, although the fern is clearly defined. The proposed method gives an ammount of detail that is between the Canny and Sobel algorithms. It shows enough details to tell what the image is without confusion. The edge image in this case retains the texture of the fern, giving it a more natural look. Details of the surrounding objects are also not lost. The image generated by the mathematical morphology uses gray-scale erosion equations introduced in Chapter 3. Its results are comparable with the proposed algorithm where both provide almost the same level of details and edges.

The second image is a drawing of a house, by Manton Reiser [43]. This image, Figure 4.9, was used since it clearly show many differences between the three algorithms. With the Canny algorithm, it can be seen that it reveals every edge. In most cases though, there are double edges. The edges are also connected in large loops. All this contributes to a very confusing scene, especially if the viewer does not know what the image is. The Sobel algorithm, on the other hand, misses many of the important details, such as the door. Thus this algorithm does not give the amount of details required for visual prosthetics. Both the proposed and the mathematical morphology images provide an intermediate level of details, both images are also very similar in their edges.

Lastly, the proposed method comes midway. It gives enough edges to recognize the image, without confusing the viewer. The more dotted look of the image closely resembles phosphenes and gives the image a more natural, realistic look. Thus, this algorithm is more suited to visual prosthetics.

Many more images were tested with the proposed method, and compared with the Canny and Sobel. They will not be discussed here, but they have all led to the same conclusion.

(a) Original Image

(b) Proposed Algorithm (Section 4.2.2)

(c) Mathematical Morphology (Section 3.6.2)

(d) Canny Algorithm (Section 3.5.1)

(e) Sobel Algorithm (Section 3.4.2)

Figure 4.8: Comparison of algorithms for the Fern Image by Jeremy C. Thomason [42]

4.4.3 Testing Noise in Bitwise Gray-scale Decomposition

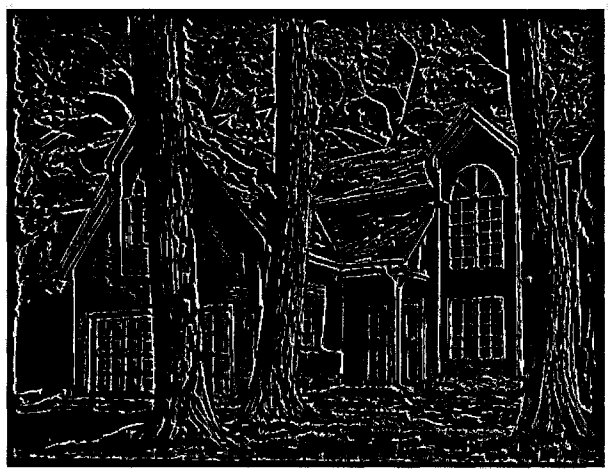
To test the proposed method of bitwise decomposition for edge detection, an image was chosen and noise was introduced to it. Once that was done, the Canny, Sobel and the Proposed algorithm were carried out on the noisy image, then compared.

The image chosen was that of a slice of cake, by Paul Johnson [44]. Figure 4.10 shows the image without noise, along with the three edge images. The conclusions drawn in the above section apply to this image, as the Sobel image misses the teacup in the background, and the mathematical morphology image misses many details.

In the next sections, two types of noise are introduced to the image, Gaussian Noise

(a) Original Image

(b) Proposed Algorithm (Section 4.2.2)

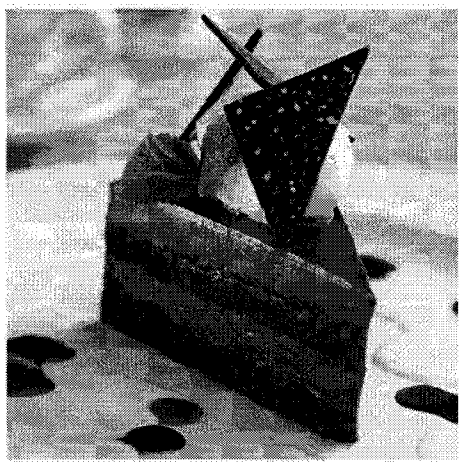


(c) Mathematical Morphology (Section 3.6.2)

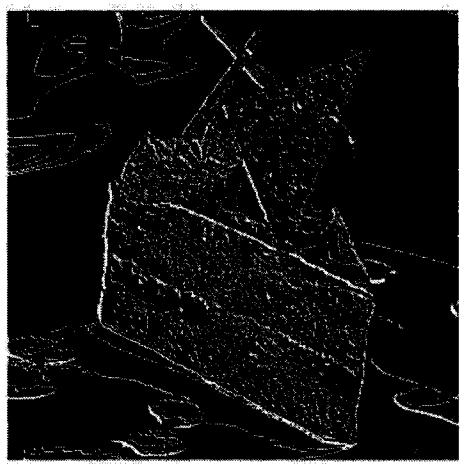
(d) Canny Algorithm (Section 3.5.1)

(e)Sobel Algorithm (Section 3.4.2)

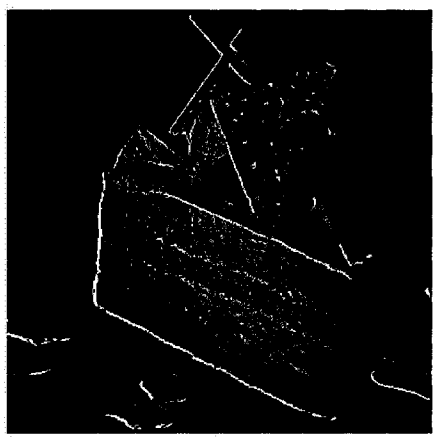
Figure 4.9: Comparison of the algorithms for the House image by D. Manton Reiser [43]



(a) Original Image

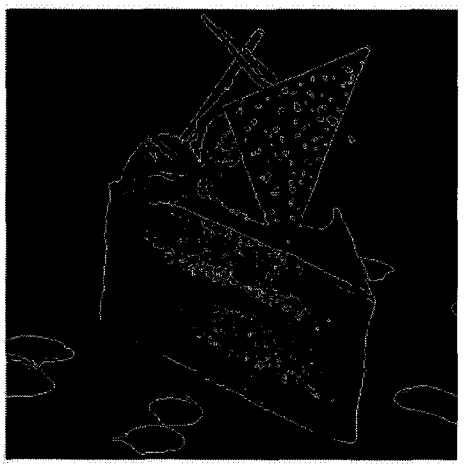


(b) Proposed Algorithm (Section 4.2.2)



(c) Mathematical Morphology (Section 3.6.2)

(d) Canny Algorithm (Section 3.5.1)



(e) Sobel Algorithm (Section 3.4.2)

Figure 4.10: Comparison of the algorithms for the Cake image by Paul Johnson [44]

and Salt and Pepper noise. These two noise types will be discussed in more detail below, they were chosen because they are the two noise types that affect DPS the most.

Gaussian Noise

Noise is present in all imagers. It is typically modelled using the Gaussian function, hence the name "Gaussian noise" [36]. A Matlab function was used to add noise to the cake image (see appendix A). Then the three algorithms are run to test their noise tolerance and shown in Figure 4.11.

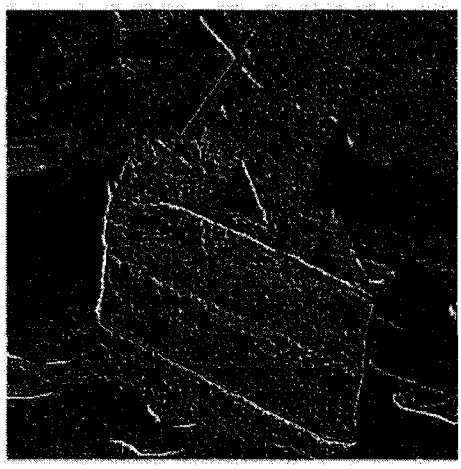
The amount of noise in the image is above normal. This is to test the worst case scenario. It can be seen that the Canny algorithm, being a good edge detector, did well. The Sobel algorithm also looks unchanged, with much less detail. The proposed method detected the noise in the background, but the general shape of the cake was unchanged. The mathematical morphology image loses much of its detail, the edges are also blurred compared to its edge image without noise.

Salt and Pepper Noise

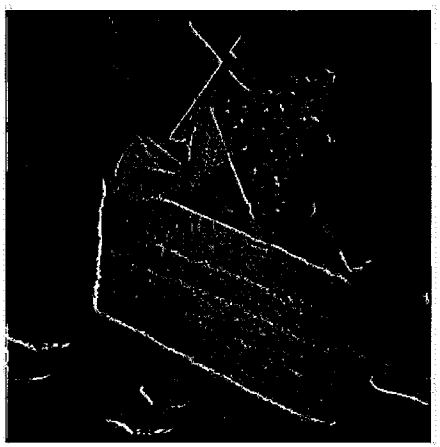
Digital pixel sensors are more prone to Salt and Pepper noise. Salt and pepper noise is mainly due to digitization or transmission errors. In general, its compared to sprinkling salt and pepper on the image. Throughout the testing of the hardware implemented array, described in Chapter 5, it was found that any error in signalling caused either a white or a black pixel output. Thus this type of noise is very important to DPS.

Figure 4.12 shows the cake image with Salt and Pepper noise added in using Matlab. All three algorithms were run on the image and the output is shown. It can be clearly seen that both the Sobel and Canny algorithms did not do well. They detected most of the Salt and Pepper noise as circles and outlined them. This was carried onto the cake image, and not only the background, which distorts the final image.The mathematical morphology image did have some problems with the noise, many dots show as edges. The Proposed method, on the other hand, did not show any problems with the Salt and Pepper noise. It did detect very few spots, but the overall image is comparable to the original. Although there are ways to deal with salt and pepper noise, the proposed algorithm inherently deals with this kind of noise.

(a)Original Image



(b) Proposed Algorithm (Section 4.2.2)

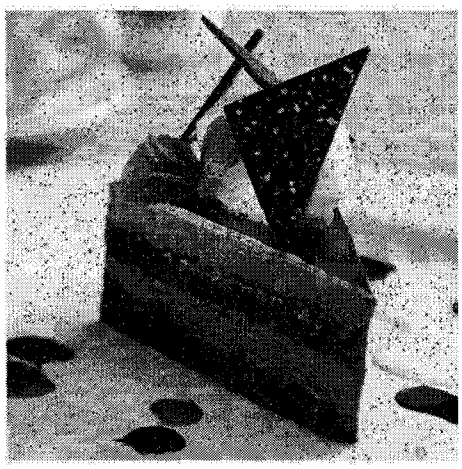


(c) Mathematical Morphology (Section 3.6.2)

(d) Canny Algorithm (Section 3.5.1)

(e) Sobel Algorithm (Section 3.4.2)

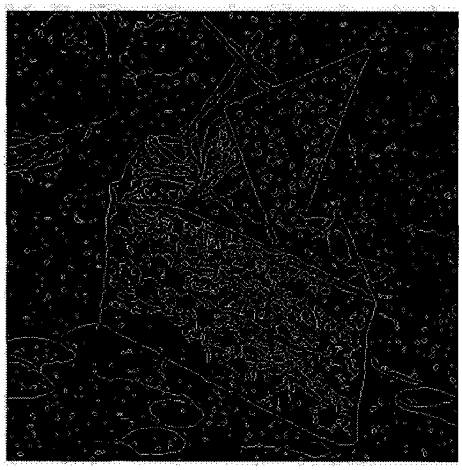
Figure 4.11: Comparison of the algorithms for Gaussian noise induced on the Cake image by Paul Johnson [44]



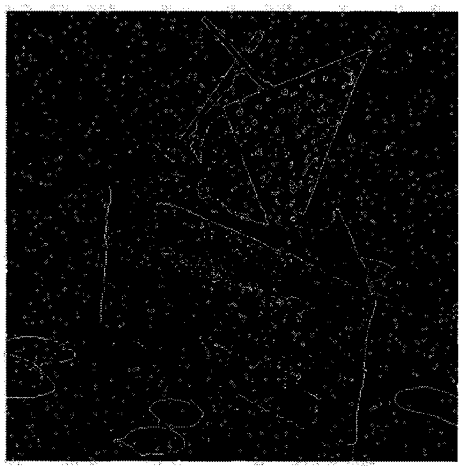
(a) Original Image

(b) Proposed Algorithm (Section 4.2.2)

(c) Mathematical Morphology (Section 3.6.2)



(c) Canny Algorithm (Section 3.5.1)



(d) Sobel Algorithm (Section 3.4.2)

Figure 4.12: Comparison of the algorithms for Salt and Pepper noise induced on the Cake image by Paul Johnson [44]

4.5 Effects of Reduced Bit Size

Typically, 8-bits are used on the pixel ADC for high-precision imaging. The design implements 8-bit resolution, but this section explores the possibility of reducing the resolution to 4-bits without while maintaining the realistic look of the resultant sketch.

Reducing the number of bits to 4 would result in faster conversion and readout times. It could also increase the fill factor by reducing the circuitry around the photodiode. Testing the reduced bit resolution starts with Figure 4.13. This figure illustrates a picture of a tulip, taken by the author, with the proposed method implemented in 8-bit mode. Figure 4.13 b) demonstrates a non-threshold result, whereas c) is a threshold version of the result.

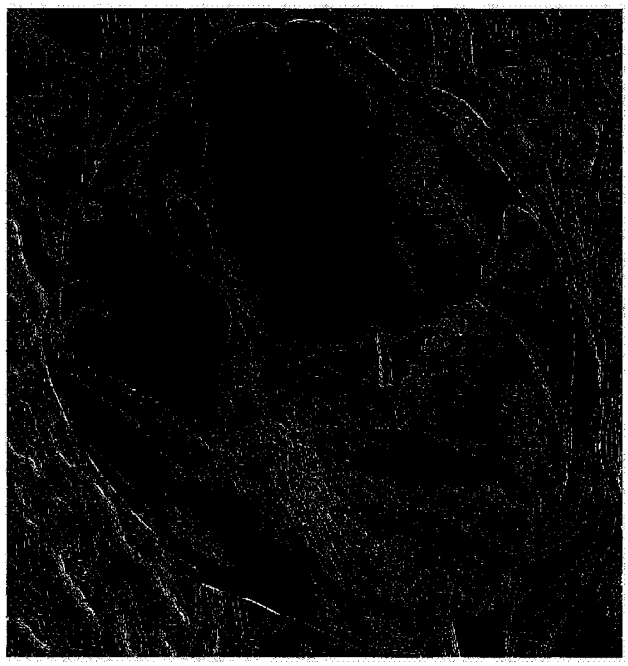
For comparison, the proposed method is carried out on four bit sizes, namely 5-bits, 4-bits, 3-bits and finally 2-bits. Since the last bits of an image are typically noise, these bit levels are achieved by shifting the last bits out, and carrying out the proposed method on the most significant bits. Figure 4.14 demonstrates the effects of the proposed method on the reduced bit size, all the images are without thresholding carried out in the end.

A 2-bit image still captures the outline of the figure, but the edges are not as clear and some detail is lost. The 3-bit, 4-bit and 5-bit capture a good amount of detail.he image the level of detail does match that of the 8-bit images, but it is comparable to that of the 8-bit thresholded image. This gives the user the choice for a faster readout by selecting to carry out the edge detection using only the most significant bits, depending on the level of detail needed. The functionality of bit-resolution can be implemented in the control unit, this would allow the user a more versatile chip.

The effect this would have on the hardware implementation is straightforward. 8-bit pixel size can still be kept, this would allow for higher precision images, if there is a needed. An 8-bit pixel can also run 4-bit edge detection by only selecting the 4 most significant bits for the erosion, and so on for the 3 and 2-bit images. To save on area though, a 4-bit pixel can be designed. It would involve a 4-bit DRAM implemented along with its 4-bit multiplexer. This should increase the fill factor once the pixel is rearranged.

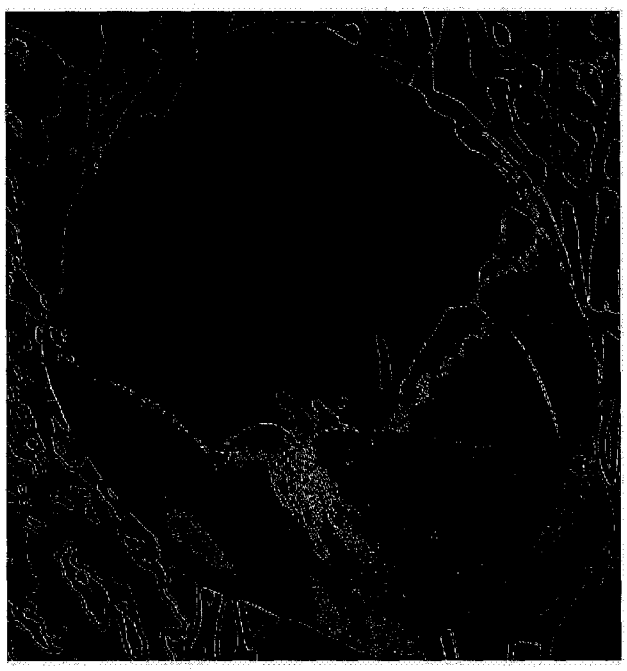
(a) Original Image

(b) 8-bit gray-scale without threshold



(c) 8-bit gray-scale with threshold

Figure 4.13: Tulip image with 8-bit edge detection carried out



(a) 2-bit gray-scale without threshold

(b) 3-bit gray-scale without threshold

(c) 4-bit gray-scale without threshold

(d) 5-bit gray-scale without threshold

Figure 4.14: Effects of varying bit size on the edge detector

4.6 Conclusion

A new method is proposed to carry out edge detection in a simple and compact manner. The implementation is based on bitwise decomposition of gray-scale images into binary images. This allows binary erosion to be carried out. Bitwise decomposition, compared to threshold decomposition allows for less binary images to be produced, and requires less hardware to implement.

It is well suited for on-pixel processing due to its simplicity. This method is implemented in mathematical morphology edge detection and is comparable with other leading classical algorithms. The Proposed method has acceptable immunity to Gaussian noise and immune to Salt and Pepper noise, making it ideal for DPS. The end result is an edge representation of the real world.

Chapter 5

Pixel System Implementation

5.1 Introduction

This chapter will cover the design of the pixel using the method proposed in Chapter 4. The implementation of each component will be covered in subsequent sections. The chapter will also cover the design layout and testing.

5.2 Pixel Design

When designing the pixel, a dilemma was reached: How many transistors can be placed while reaching the goal of increased fill factor?

Several designs were reviewed, the first being the addition of all the edge detection functionality into the pixel. This approach would need two AND gates, a subtractor, a comparator, several latches and other circuitry. These components would yield a large transistor count. From the literature review, this means a fill factor of less than 10%. The second design is to incorporate the erosion and part of the edge detection. This approach would also give a fill factor of around 10% or slightly more.

The last approach is to simply put the erosion on pixel, the rest can be done off pixel. This approach would give a higher fill factor, and it is the most versatile. Literature on this matter reports that the pixels implemented have partial image processing abilities, yet, they yield a FF of 10% or less [12] [13]. In this approach, the FF is increased and the off pixel control is simplified.

Figure 5.1 shows a typical pixel array. The pixel array carries out partial image processing, and the results are then sent to the processing unit. The pixel itself requires many control lines, and thus a complex processor to control it; in some cases, a SIMD (Single Instruction, Multiple Data) and integer pipeline [12]. Since the image is not fully processed, the pixel array also requires another processor to finish off the image processing. This makes the pixel less portable and consume more power.

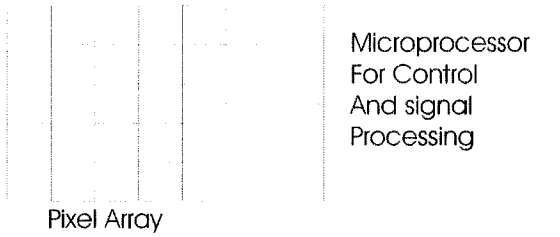


Figure 5.1: Pixel arrays typically need microprocessors

In the proposed pixel design, shown in Figure 5.2, the pixel array has a higher FF compared to the designs mentioned in the literature review. Due to its simplicity, it does not need a complex control unit, as will be seen in the following sections. The output of the pixel array is both the image and the erosion needed to carry out edge detection. Thus the control unit has to carry out the remaining processing in the edge detection algorithm. Other designs reviewed in the literature carry out binary morphological operations, in this design, gray-scale morphology is applied, which allows to capture the image in gray-scale, capturing more details compared to binary.

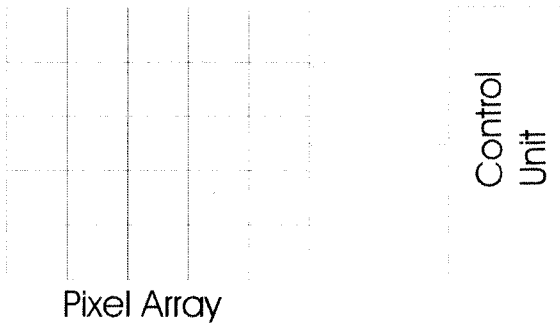


Figure 5.2: Proposed pixel array only needs a simple control unit

The next sections will cover the components in the design of the pixels. The design

covers the pixel itself and the control unit is implemented in verilog; the final stages of edge detection are done in Matlab for comparison with Matlab simulations. But first, a brief overview of the pixel operation is presented, and more details will be covered in the testing and results chapter.

5.2.1 Pixel Operation

Each pixel has its own ADC. The ADC starts with the photodiode that once stimulated by light, generates a current. The current is integrated and the voltage is fed into the positive input of the comparator. An analog ramp, running in sync with a counter, is placed on the negative port of the comparator. Once the voltage from the photodiode crosses that of the ramp, the output of the comparator goes from high to low. This output is tied to the write signal of the on-pixel memory, in the form of a read/write DRAM. An 8-bit counter is connected to the pixel memories IO lines. Once the write signal is low, the value is latched into the DRAM. Now the erosion processes takes place, bit by bit. Starting from the least significant bit, the memory is read and placed into the erosion circuitry. It is then latched and read into the control unit.

The operation of the pixel may seem simple. However, keeping the number of lines to a minimum, for minimum space, requires very careful timing of the signals and a careful design.

5.2.2 Signal Control Design

The design of the pixel is chosen such that the fill factor is maximized. In doing so, the pixel uses many shared buses in an attempt to minimize the number of control signals. This creates the need for careful design of the control unit to avoid any race conditions and conflicts on the buses. The array is arranged such that the read and write control signals are generated column by column. The image and the resulting erosion is read row by row. Figure 5.3 gives a simplified overview of the operation of the array.

In more detail, to control the pixel the following steps are taken:

1. Restart State: Set Vgp, the photodiode reset signal, to 1. All other signals are reset to zero.

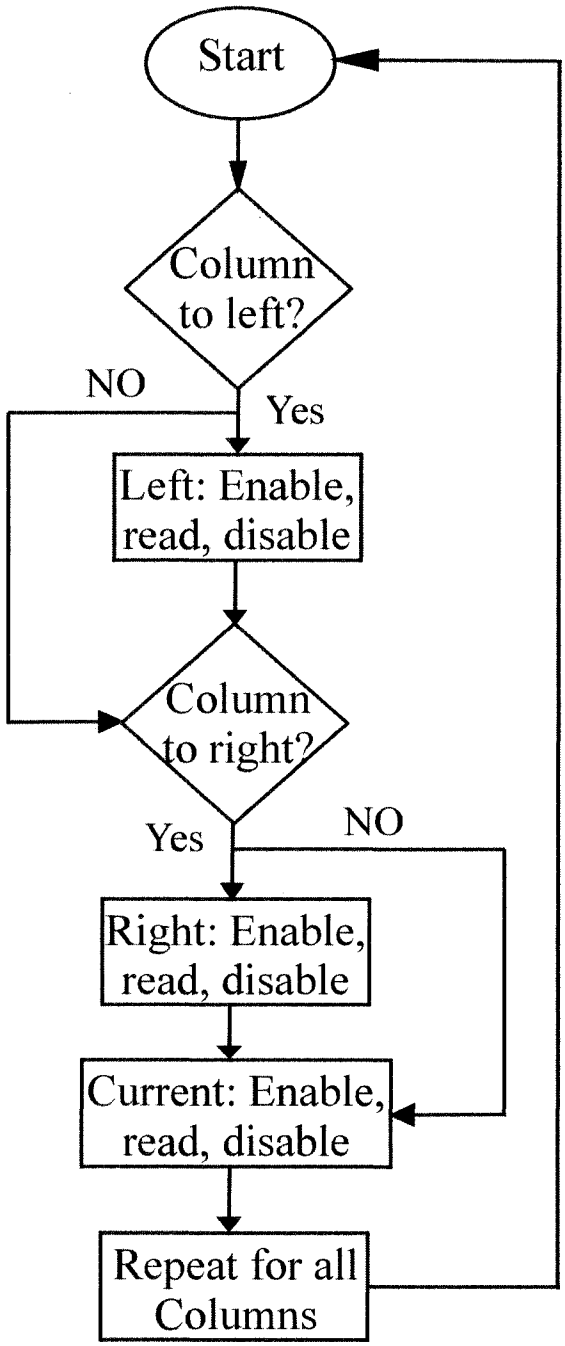


Figure 5.3: Flow chart of the controlling the array

2. Photodiode Current Integration: Turn Vgp to zero. Enable the bias and power-down block. In this step, the logarithmic mode can be activated by enabling Vgn. The logarithmic mode is used for applications needing a wide dynamic range.

3. ADC: Initiate the Analog Ramp and the Graycode counter.

4. Start of Erosion: To start the erosion, the image is read from the memory. Since the pixel takes the values from its neighbours, the column on its left and right have to be enabled, which bring up 3 different scenarios. But first, the bias and power-down block must be turned off to conserve power.

   The Left Edge (Column one): Step one is to enable the read, output latch and tristate buffer of the second column. Once the value is latched, disable its read and latch only. The second step is to enable the read, all latches and tristate buffer of the first column. Image values will be read and processed. finally, disable all signals.

   Middle Pixel: First step is to enable the read, output latch and TS Buffer on the column to the left of the current column. Once the values are latched, disable the latch and the read only. Next, enable the read, output latch and TS buffer of the column to the right of the current one. Then disable the latch and read only. Finally, enable all the latches, tristate buffers and reads of the current pixel. Once processing is done, disable all the signals.

   The Right Edge (The last column): The right edge is very similar to the left edge scenario, but when it comes to enabling the columns, instead of the first and second columns, its is the last and before the last columns respectively.

5. Now that all the values have been collected, the remainder of edge detection can be carried out.

The above steps are implemented in Verilog code. The remainder of the edge detection is implemented in Matlab. One of the reasons is, speed, as simulating the Verilog code takes much longer than a Matlab code. Also, the results from the array can be compared easily with results form Matlab run on the same image. Input/output commands were used in the Verilog code to save the output of the array and to input the image into the array.

5.2.3 Control Unit Design

The testing of the array involves writing a Verilog code to control the signals. Shown in Appendix B, the code is written in a test bench format. This style does not allow the code to be synthesized. Thus, after the functionality of the array is tested and the timing and order of signals known, a new all inclusive control unit is written.

This control unit is for a 16x16 pixel array containing both the photodiode and processing element. In order to synthesize the unit, a finite state machine was developed as shown in Figure 5.4.

The clock, fed into the control unit, runs at 500 MHz. A counter is kept to keep track of time. This counter is called "cnt" in the code and in the diagram above. The state machine stays in the reset mode for 6 ns, that is, when the counter reaches 3.

The next state is the integration of the photocurrent. This state takes about 188 ns. Once cnt reaches 97, the machine moves to the next state, which is the analog-to-digital conversion. This state is timed for 40 μs. The other states mimic the for loops in the test-bench code.

Since the states are triggered on positive edges of the clock, consecutive states are 2 ns apart. The first state in the loop is called S3. S3 reads in the values from the column to the left of the current pixel column. The next state S4, reads the column to the right. S5, reads in the current pixel column, and moves on to state S6. In this state, the values are read into the memory. The last state in the loop is S7. In this state, all signals are de-asserted, and the counters are incremented. Every time the counter 'j', the inner loop, reaches 16, the outer loop counter 'i' is incremented, and 'j' is reset. As long as 'i' is less than 8, the machine will set the next state as S3, reading the columns one at a time. Once 'i' reaches 8, the loop exits and the next state is set to reset, S0.

The control unit can be synthesized, but the synthesis tools available do not support the CMOSP13 design kit. The generated schematic can then be implemented manually or as an alternative through one of Xilinxs FPGA tools.

5.2.4 The Photodiode

When simulating the pixel, a good photodiode model is needed. Typical models include a current source and a diode. This does not take into account the capacitances and resistance

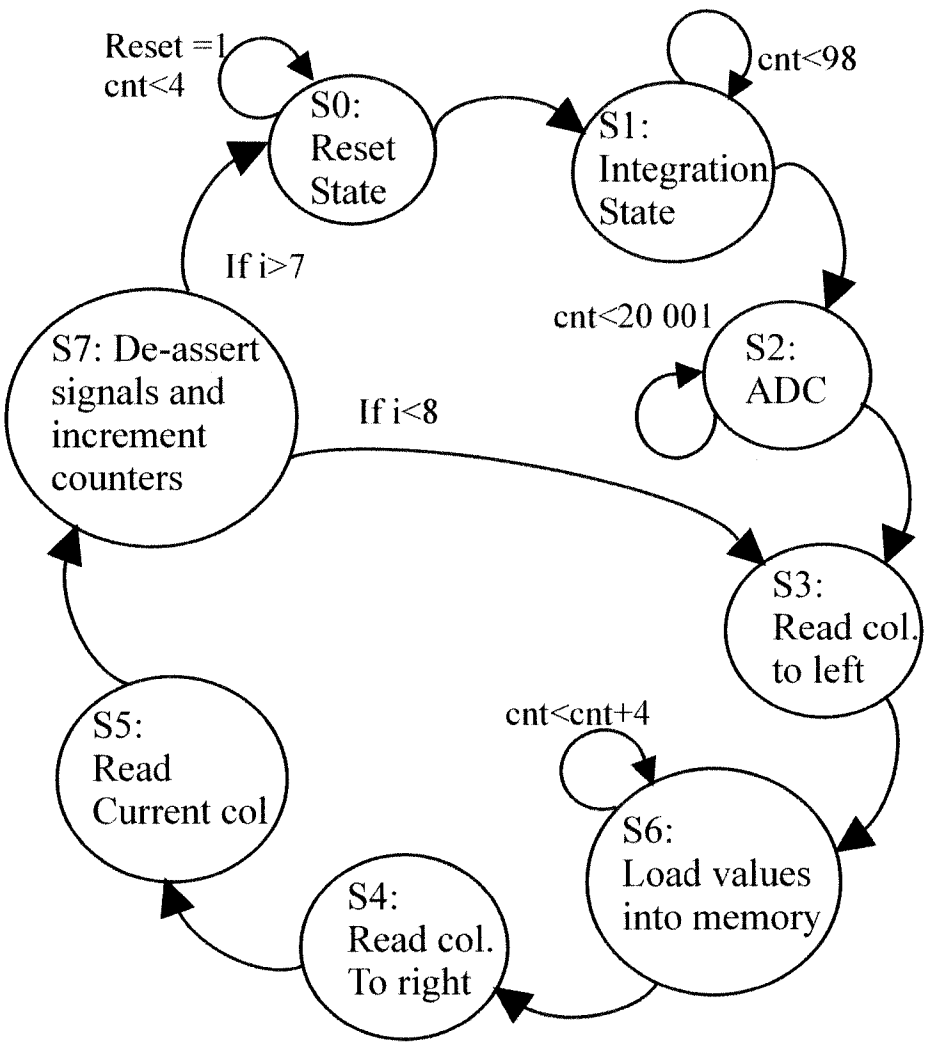


Figure 5.4: Finite state machine used for the control unit

of the photodiode, and thus does not provide accurate results.

A more accurate model involves a current source, diode, capacitance and three resistances, as shown in Figure 5.5. The capacitance models the depletion capacitance of the diode at reverse bias. The value of the capacitance is calculated from the size of the photodiode, which will be discussed in more details in the layout section [46].

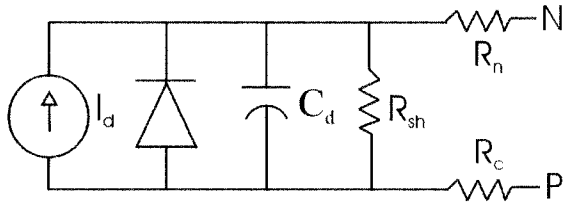


Figure 5.5: Schematic of the photodiode model

The large resistance is the shunt resistance, typically in the mega ohms range. The two small resistances model the parasitic resistances at the cathode and anode of the photodiode. Thus, this model is best suited for simulation. The values used for the components are shown in Table 5.1. By varying the current source, the simulated photodiode current is changed. The capacitor value, C_d, is calculated after the final capacitor size is decided, the calculations are presented in the section on layout. The resistor values are typical for a photodiode, they are fine tuned to the capacitor value.

Table 5.1: List of the Values used in the Photodiode Model

Component	**Value**
Current, Ipd	Variable
C_d	0.08pF
R_{sh}	1MΩ
R_c **and** R_n	15Ω

5.2.5 Photodiode Reset Circuitry

The photodiode is connected to two transistors. One of the transistors is used to reset the diode which is an NMOS. Typically a PMOS is used, but in this case the NMOS would provide a better fill factor, without affecting the functionality. The reset transistor is controlled by the line called V_{GP} [1].

The second transistor is the shutter transistor, used to give the pixel two extra operating modes, linear integration and logarithmic mode. The linear integration mode provides a high signal to noise ratio and high swing outputs. In this mode, the second NMOS transistor is not needed [1].

The logarithmic mode is useful for applications requiring a wide dynamic range. In this mode, the second NMOS transistor needs to be switched on in weak inversion. This makes the photodiode voltage follow the relation in equation (5.2.1), where I_o is a constant, κ the Boltzman Constant, (1.38x10^{-23}J/K) and q the Electron Charge (1.6x10^{-19}C) [1].

$$V_{photo} = V_d - \frac{\kappa T}{q} \ln(\frac{I_{photo}}{I_o}) \tag{5.2.1}$$

Analog to digital conversion is carried out as usual after the initial photodiode voltage is read, weather its the logarithmic mode or the linear integration mode.

5.2.6 The Comparator

The comparator was implemented using three stages. The first being a differential gain stage, the second a single-ended gain, and the last a CMOS inverter to saturate the output voltage [3]. This configuration allows for low power and small area implementation shown in Figure 5.6.

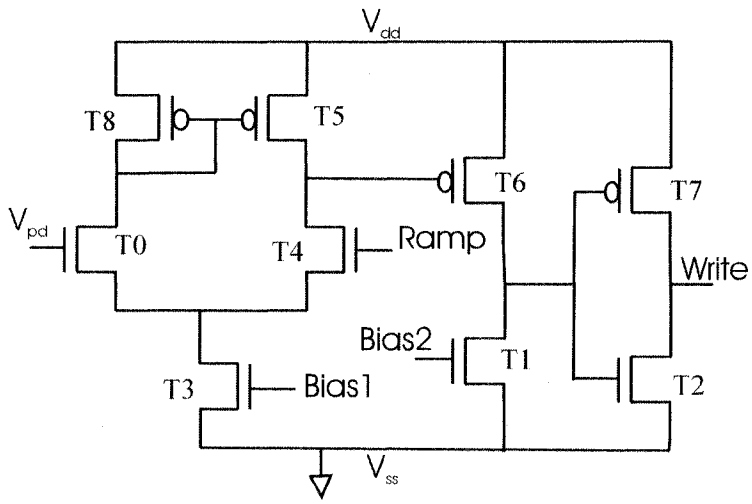


Figure 5.6: Schematic of the comparator

The sizes of the transistors used are shown in Table 5.2. The transistor length are at the minimum values allowed by the technology parameters. The widths are varied to achieve the required result.

Table 5.2: Transistor width used for the comparator

Transistor	Value nm
Differential pair	
T0,T3,T4,T5,T8	280
Single ended gain	
T1	280
T6	400
Inverter	
T2	280
T7	320

5.2.7 3T DRAM Unit

The structure used for one bit of memory is the 3T DRAM. This structure provides small area and fast readout. Due to its simplicity in design and operation, the 3-T DRAM cell is the cell of choice in application-specific ICs, such as this one [47].

Each cell has a shared input/output line, and a read and write control line [3]. For a compact design, the transistors' sizes were all made equal, with a length of 200 nm and width of 280 nm. The 3T DRAM inverts the output, thus, if 1 is stored, 0 would be read.

To combat off-currents, the input output lines had to be held at Vdd, which was 1 V. The lines were tied to weak PMOS pull-up transistors. Where each row of the array had a set of 8 pull-up transistors, tied to the 8 input output lines of the 8-bit DRAM.

5.2.8 8-Bit DRAM

The 8-bit DRAM consisted of placing eight of the one bit 3T DRAM cells. With their read and write lines tied together. The IO lines were labeled IO0 to IO7.

5.2.9 Pass Transistor Logic Multiplexer

The multiplexer was implemented using NMOS pass transistor logic, also called a tree-based decoder [47]. This design is compact and easy to implement on pixel. All transistors had minimum sizes.

The tree decoder was used due to its reduced transistor count compared with other decoders. For a tree decoder, the number of transistors required for a 2^k-input decoder is given by equation (5.2.2). For example, an 8-to-1 decoder requires only 14 transistors. Speed is not an issue since there are only three serially connected pass transistors. Thus the transistors are not tapered [47].

$$N_{tree} = 2^K + 2^{K-1} + ... + 4 + 2 = 2(2^K - 1) \tag{5.2.2}$$

The row decoder, used to read the values from the array, would also be implemented with a tree structure. There would be three 16-to-1 decoders. One for the pixel value and two for the erosion outputs. Each would require 30 transistors. To overcome the delay, buffers are inserted that are tapered where the transistor sizes increase from bottom to top.

5.2.10 Two, Three and Four Input AND gates

The middle pixels require only four input AND gates, but the edge pixels need both a two and a three input AND gate, whereas the corner pixel needs only a two input ANDs. This is due to the fact that these edge and corner pixels do not have all their neighbours. The AND gates were implemented using CMOS design [47], [48]. All AND gates were implemented with a CMOS NAND gate followed by a CMOS inverter.

5.2.11 Pass Transistor Latches

Because of its compact design, pass transistor logic has been used wherever possible. The latches used at the output of the pixels is shown in Figure 5.7. At the input, the transmission gate allows the value of D to pass only when the CLK signal is high. Once the CLK signal is low, the transmission gate at the output is on, and that at the input is off. This forms a loop latching in the value of Q [48] [49].

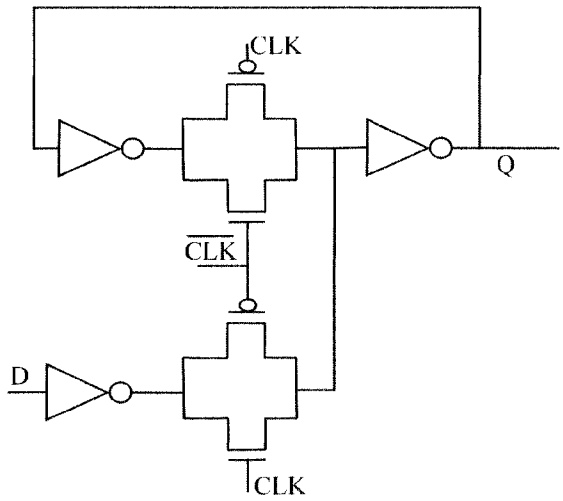


Figure 5.7: PTL latch schematic

5.2.12 Pass Transistor Tristate Buffers

The PTL tristate buffer has two major advantages over the CMOS implementation. First of all, it has two less transistors, second, it is non-inverting. The tristate buffer consists of a PMOS and NMOS transistor controlled by a set of complementary signals.

5.2.13 Bias and Power-Down Circuits

Although they have not been implemented in layout, the bias and power down circuitry has been implemented and used in the testing of the pixel.

As shown in Figure 5.8, the circuit consists of a current mirror, transmission gate and a pull down transistor. This provides the ability to power down the comparator when not in use, thus conserving energy and reducing power consumption [3].

5.2.14 Proposed Row and Column Decoders

There are two approaches to controlling the pixel array. The first involves the use of a column and row decoder. The second is the use of Verilog code to model the signals. The second approach is used in testing the pixels. But a column and row decoder are designed, where the column decoder and the row decoder are a simple expansion of the 8-bit multiplexer to 16 bits [47] [48] [49].

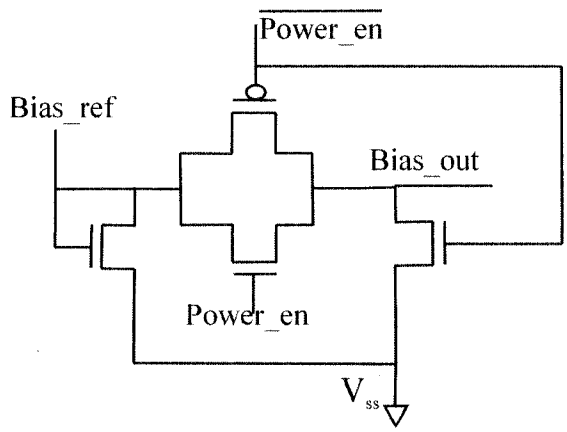


Figure 5.8: The bias and power down circuit

5.2.15 Processing Element Implementation

The processing element is composed of the 8-bit DRAM, the multiplexer, whose output is connected to a latch, then the tristate buffer. Two AND gates carry out the erosion, connected to a latch and tristate buffer. The processing element is shown in Figure 5.9. The corner element uses the 2-input AND, the edge element both the 2-input and the 3-input AND.

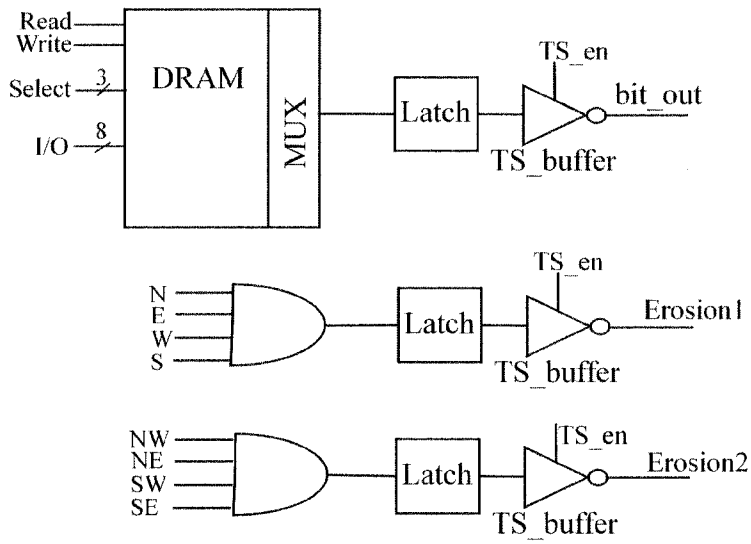


Figure 5.9: The processing element

5.3 Pixel Layout Implementation

This section covers the layout techniques of the pixel. The entire pixel is discussed in the end, where the fill factor, pixel abutment and other factors are discussed.

5.3.1 Photodiode

The photodiode size and shape are important factors in its design [50]. In this pixel, a simple square shape was chosen. To properly choose the size of the photodiode, several considerations must be made. First, the larger the photodiode, the larger the pixel, thus giving lower resolution. Also, a larger photodiode means a larger capacitance, resulting in a slower pixel. The advantage of a larger photodiode is a better fill factor.

Several sizes were considered ranging from 5x5 μm to 8x8 μm. The capacitor C_d was calculated using the following formula: C_d = Area $\times$ C_{jarea}, where C_{jarea} is the capacitance per square μm. Thus, the values of the capacitances were calculated, and placed into the photodiode model. A simulation is run with the comparator to test its output. The conversion voltages were very similar, thus a photodiode of size 8x8 μm is chosen, with a capacitance of 0.08pF, which is within photodiode capacitance limits. Figure 5.10 depicts the photodiode current versus the analog ramp voltage at which the write signal switches.The converter has a range of 45 mV to 910 mV.

5.3.2 8-Bit DRAM

The 8-bit DRAM is a stack of eight one-bit DRAM units. The design gives it a long slender shape, ideal for placement on-pixel, shown in Figure 5.11.

5.3.3 Complete Pixel Layout

The layout of the complete pixel has to be made as tight as possible. The first step was to make the AND gates, latches and tristate buffers connected into one module. Then the latches and the DRAM can be arranged around the photodiode. The best fill factor was achieved when the DRAM is placed to the left of the photodiode, and the the latches to the right. The rest of the components are placed on the top. The ground line is placed around the pixel, such that, once placed side by side, they can connect. The overall pixel

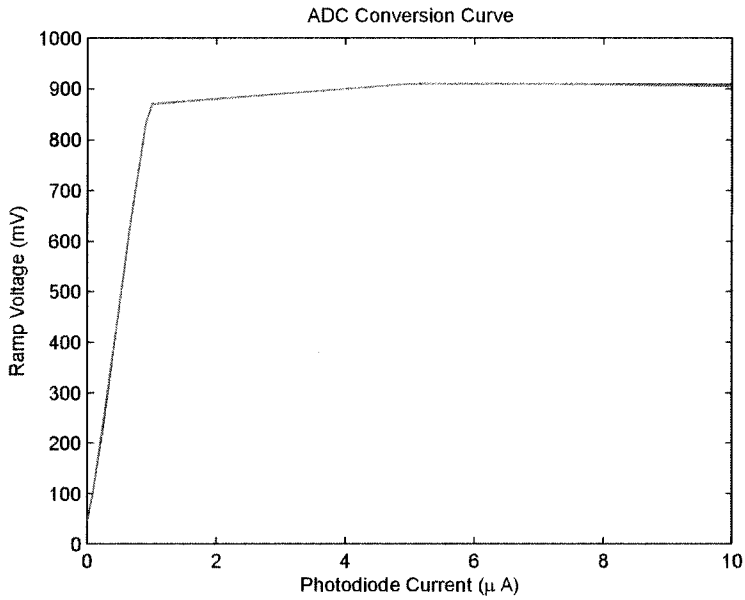


Figure 5.10: Photodiode current conversion curve

size is 17 x 17 μm and the fill factor achieved is 22%. The entire layout is shown in Figure 5.12. The corner pixel and the edge pixel are shown in Figures 5.13 and 5.14 respectively.

5.4 Pixel Array Test Cases and Results

This section introduces the testing techniques used in testing the proposed design. The results are also presented in this section with a summary of the pixel array. Each component in the design is tested separately to verify its functionality, the results presented will only be those of components connected together.

To make an array of pixels and simulate it in CAD tools will not allow the use of the photodiode. The reason being many instances of the photodiode would have to be created such that they would not all have the same current. This means that a 16x16 array would need 256 instances.

The solution is to run several tests. The first is to verify the functionality of the photodiode, reset circuit and the comparator. Then, the processing element is tested along with the DRAM. Finally, simulations are run on the pixel including the photodiode.

Arrays are then created to test the functionality of the design as a whole. These arrays

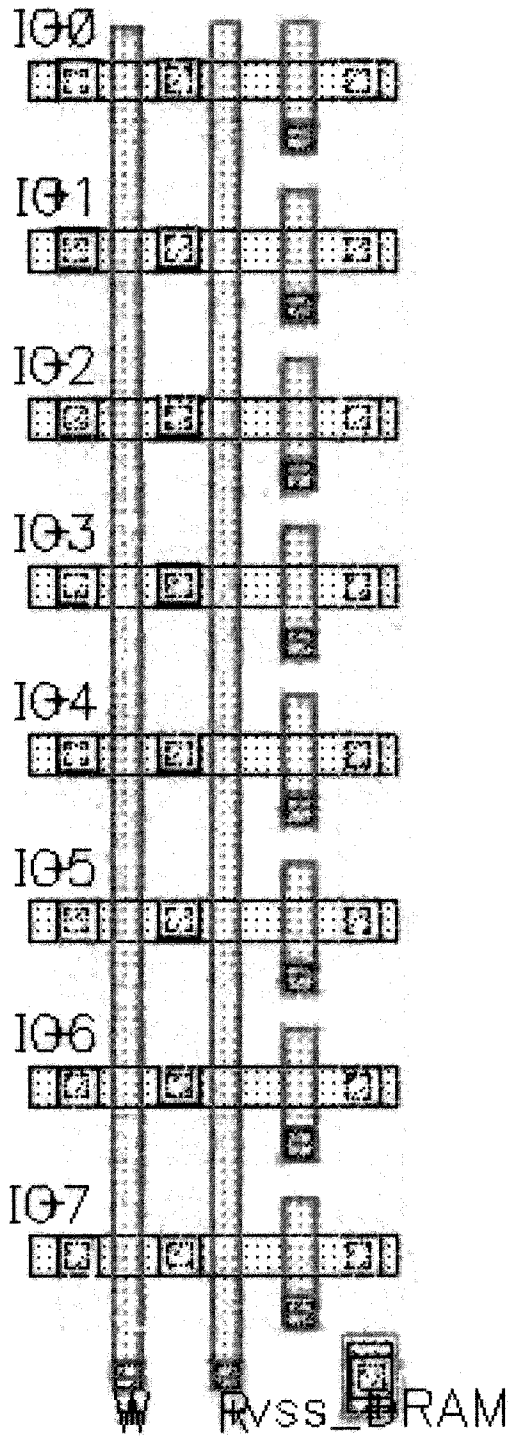


Figure 5.11: Layout of the eight bit DRAM

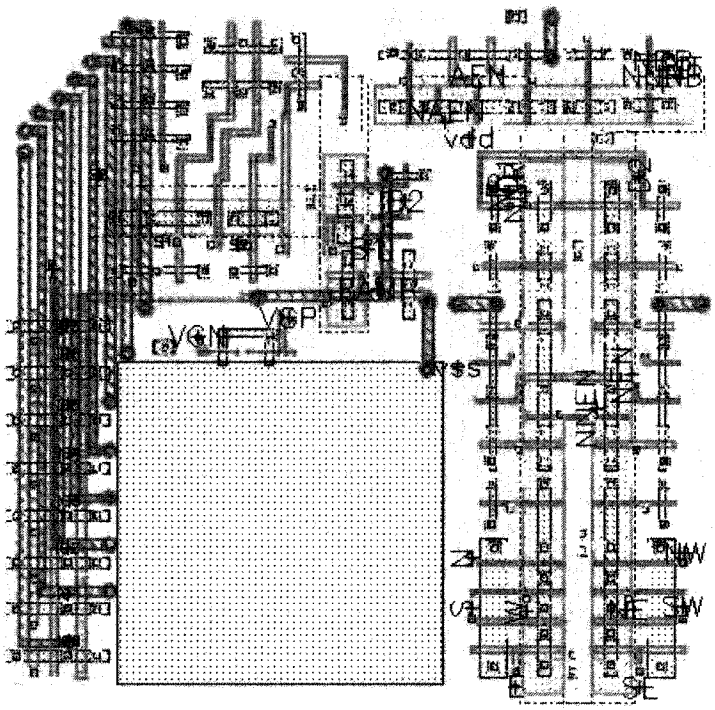


Figure 5.12: Layout of the complete pixel

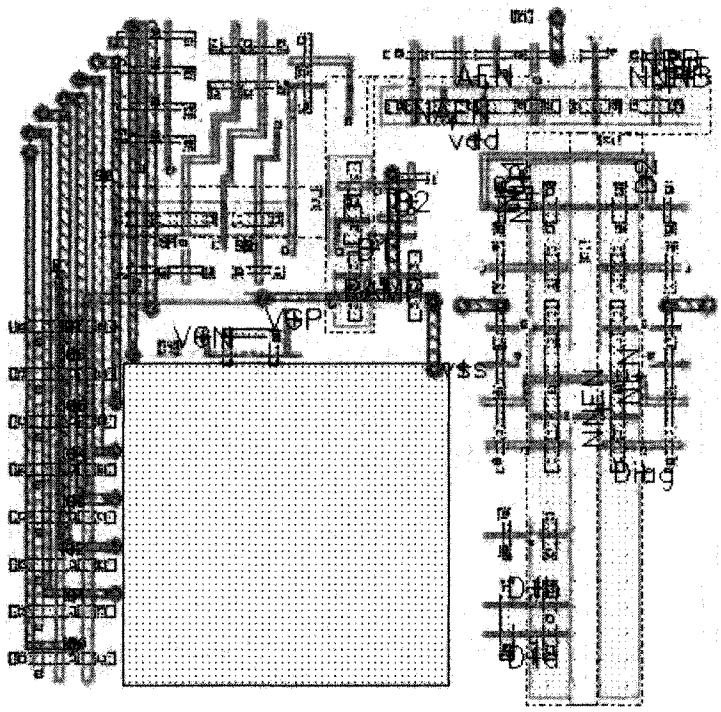


Figure 5.13: Layout of the complete corner pixel

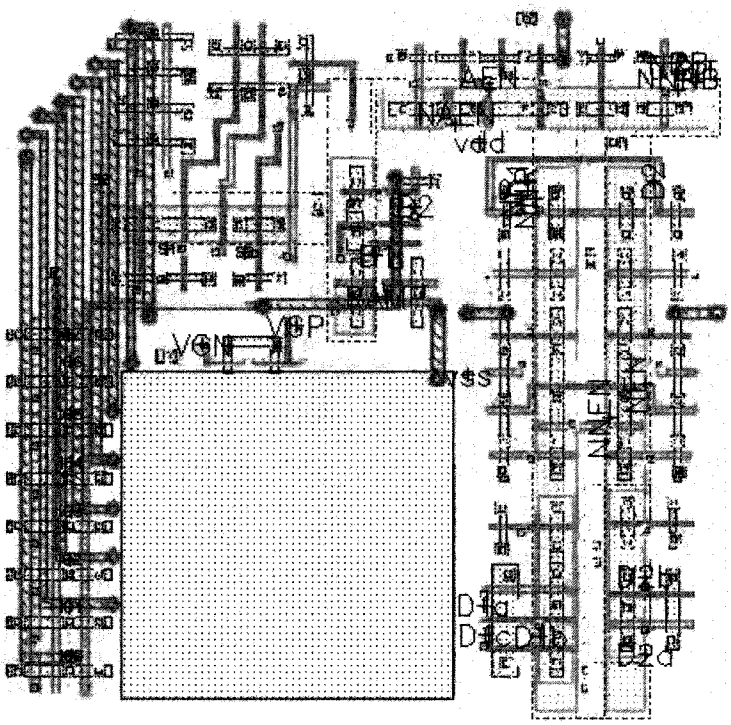


Figure 5.14: Layout of the complete edge pixel

do not include the photodiode and comparator, due to the above mentioned reasons. The arrays are loaded with an image from memory, run, then the results collected and compared. The components are all designed to run at a voltage of 1 V.

5.4.1 Testing of the Photodiode and the Comparator

Once connected, the photodiode and the comparator are simulated under varying current values. The current range for the photodiode is: 25 nA to 5 μA. This means that any current from the photodiode less than 25 nA will not affect the write signal. Thus a dark current of less than 25 nA is acceptable.

Figure 5.15 shows that once the output voltage of the photodiode crosses the analog ramp, the output of the comparator is taken from high to low. This signal is used as the write signal to the DRAM, thus once low, it will latch in the value of the counter at the DRAM IO.

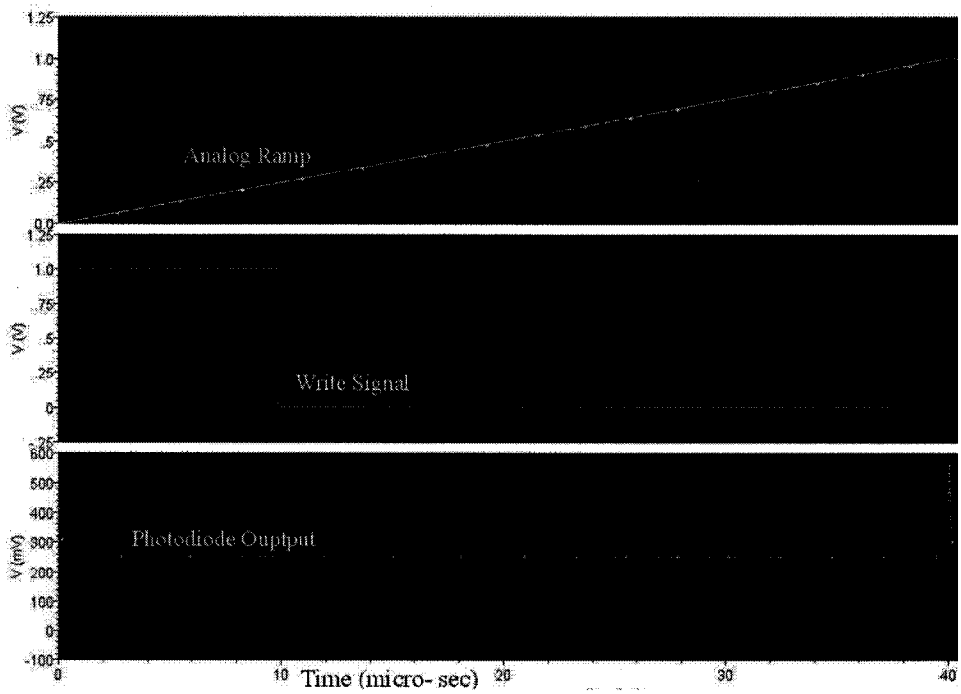


Figure 5.15: Test results of the comparator and photodiode

The comparator has a range of input current for which it will switch. The maximum current is 5μA and the minimum current as 25nA. These values are used to estimate the

dynamic range of the design. By using the equation introduced in Chapter 2, the DR comes to 46 dB, which is well within the range of typical DPSs.

5.4.2 Testing the Processing Element

The processing element and the DRAM are tested without the photodiode. Verilog code is written to control the unit and verify it. The code consists of several registers holding values for the pixel and its eight neighbours. At the start of the code, the DRAM is loaded with the pixel value (which can be changed in the code). The code then iterates to read out the value latched in the DRAM, load the neighbour values and the output of the erosion, all bit by bit. By knowing the values of the neighbour bits set in the code, the output of the erosion can be checked.

Schematic Results

Results from the simulation are shown in Figure 5.16. The Verilog code had a pixel value of 55 or 37H. The neighbour values are FFH for pixels 2,4,6 and 8, corresponding to SE2. The values for SE1 are, 37H, 37H,14H and 3C.

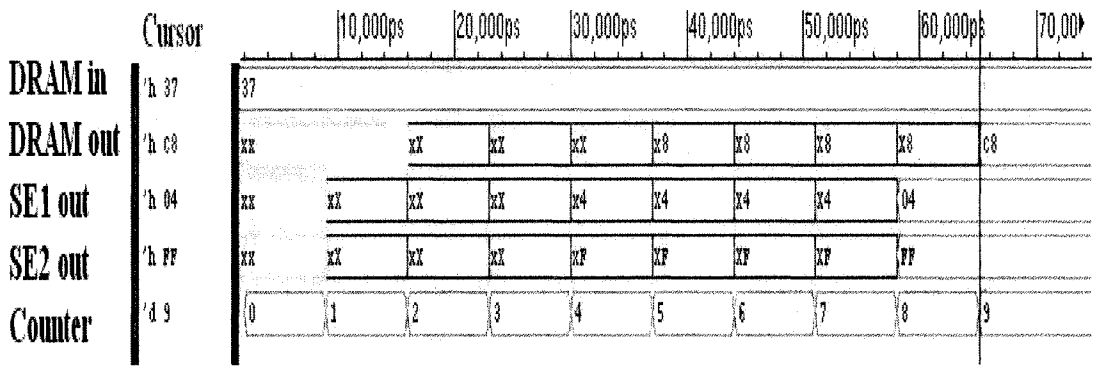


Figure 5.16: Test results from the processing element

Thus the output of the pixel DRAM should be the inverse of 37H, which is C8H. The results from the SE1 erosion should be 04H and SE2, FFH. Figure 5.16 shows that the PE is working as expected. The power dissipation was calculated using the RMS current, which it comes to 25 μW.

Extracted Results

By using the AMS hierarchy editor, the view used for simulation was set from schematic to extracted. Once the simulation was run, the results were identical to the above. The only difference was a slight decrease in power consumption at 8.73 μW.

5.4.3 Results from the Complete Pixel Tests

This test makes sure that the pixel works all together including the photodiode. A current value is loaded into the photodiode. A gray code counter is created in Verilog, another control unit is written in Verilog code to control the tristate buffers, the pixel and the bias and power-down unit. Tristate buffers are placed between the counter and the IO lines of the pixel . Once all the components were connected, the simulation is run.

The first results extracted from the simulation were waveforms indicating how the write enable works. This is needed to see the value of the counter at the point in which the write is disabled. Figure 5.17 shows that the write becomes low at a counter value of D4H. Now, to make sure that the value was truly latched in, the output of the pixel has to be checked.

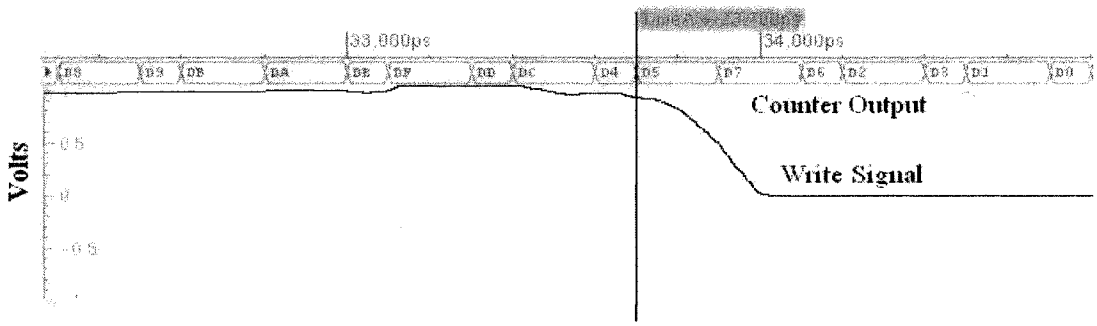


Figure 5.17: Test results of the complete pixels write signal

The values of the neighbour pixels are the same as those in the PE test, except for SE2, where they were all zeros. Thus, Figure 5.18 shows that the output of the pixel is 2BH, for SE1 is 04H and for SE2 00H. The inverse of D4H is 2BH. Thus the pixel has been verified to work. No extracted simulations were run since the current from the photodiode can not be simulated in layout. The overall power consumption of the entire pixel was 15 μW

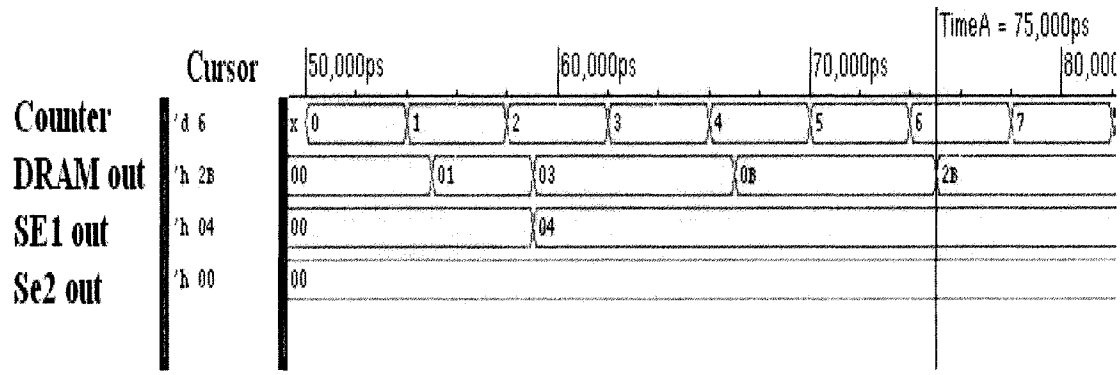


Figure 5.18: Test results of the complete pixel

5.4.4 A 4x4 Array

A 4x4 Array of PEs is designed to test the functionality of the proposed erosion method. A control unit written in verilog sends the signals to the array, and receives the results. The results are then written to file for comparison. To load the array with an image, a 4x4 snapshot of a gray-scale image is taken, using Matlab, then loaded onto the array using verilog code. The code reads in the file, then enables the read signals one column at a time. Once the values are stored, the erosion can begin.

Schematic Results

The most important result from the array is the erosion output, and weather it results in the desired edge image. After running the simulation, the remainder of the edge algorithm is run and compared to a Matlab simulation of the same image.

Since a 4x4 would not yield an image in Matlab, the original image is shown in Table 5.3. The output of the array is identical to the Matlab output where both are shown as Table 5.4.

Table 5.3: Image used in the 4x4 array test, all values are hexadecimal

76	5E	5C	5E
68	76	7A	98
A0	A7	A6	A0
9F	A2	A4	A1

Table 5.4: Resultant edge image in the 4x4 array test

0	0	0	0
1	1	0	1
1	1	1	1
0	0	0	0

Thus, an edge is detected as expected and the results match the desired output. The overall power consumption of the array came to 73 μW.

Extracted Results

The layout for the middle, corner and edge pixels are extracted. These are placed as the views in the hierarchy editor and the simulation is run. The results of the simulation are identical to that of the schematic results. The only change is a slight difference in the power consumption, coming to 75 μW. The energy expended for both the schematic and the extracted comes to 0.018 nJ. The simulation takes 243 ns to complete. Thus the functionality of the array has been verified.

5.4.5 An 8x8 Array

A 4x4 is not a practical array size for an imager, thus larger sizes have to be tested to observe the scalability of the design. The simulation run time is 643 ns. The delays were increased to accommodate the simulator.

Schematic Results

An 8x8 image is loaded into the array, and the simulation is run. The resultant image is shown in Figure 5.19. This image matches that of the Matlab simulation. Clear crisp edges are defined in the figure. Although the image does not make sense, it shows that the array is working well. The power consumption for the schematic run is 0.192 mW, where as the energy is 0.123 nJ.

Extracted Results

The extracted results match that of the schematic. The power consumption comes to 0.114 mW. Energy consumption is 0.073 nJ. The current waveform is more stable in the

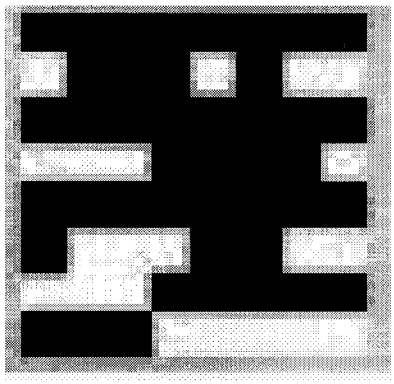


Figure 5.19: Test results of the 8x8 array

extracted simulation.

5.4.6 A 16x16 Array

The final array test is that of a 16x16 array. Once again, a 16x16 image is loaded. Figure 5.20 shows the approximate area that the image is taken in. The simulation run time is 1.559 μs.

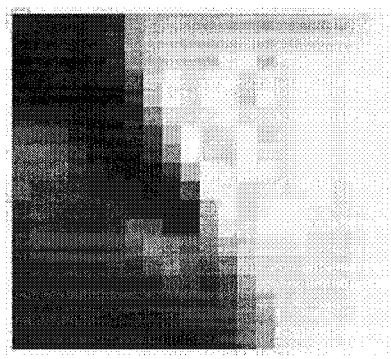


Figure 5.20: Image used in the 16x16 test

Schematic Results

The Matlab simulation results are shown in Figure 5.21. These results differ slightly from the simulation results and from the array. The power dissipation comes to 0.38mW where as the energy consumption comes to 0.52 nJ

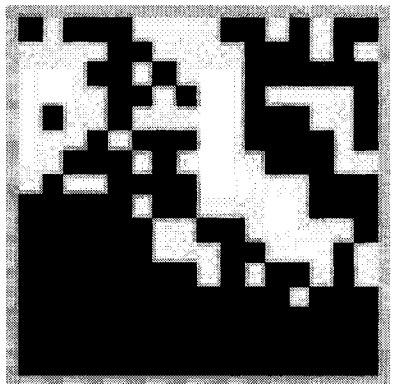


Figure 5.21: Matlab simulation results for the 16x16 array

Simulation results are shown in Figure 5.22. The difference in results are not significant, but can be attributed to simulator timing issues. With the increase in array size, the simulator slows down and has caused problems. But the extracted results as seen next, achieve correct results.

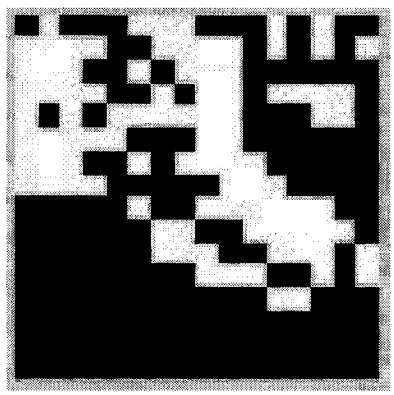


Figure 5.22: Schematic level simulation results for the 16x16 array

Extracted Results

The extracted results match that of the Matlab simulation, shown in Figure 5.23. The reason for the difference between the schematic and extracted results can be attributed to the way that the layout is designed. In the layout of the pixel, there are no hierarchial elements, the design was flat. In comparison, the schematic design had instances of every component. The deeper the hierarchy, the longer the simulation will take, making the

simulator run out of memory before the results have stabilized. Extracted results simulate the results that would be achieved on silicon, thus they are more valuable. Since the layout and extracted pixels are flat, meaning no hierarchial elements, the simulator runs on time and produces the correct results, proving that the array does indeed work properly.

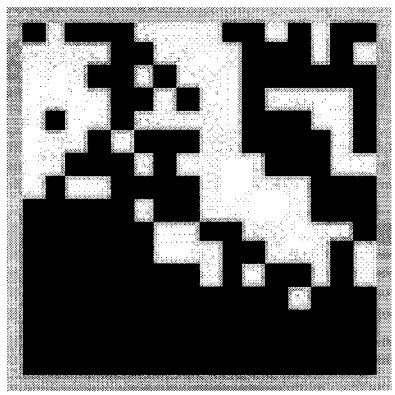


Figure 5.23: Extracted level simulation results for the 16x16 array

The power dissipation does doubles from the 8x8 array. This is due to the activation of one column at a time, where the other columns would be in an idle state. Thus the power dissipation does not quadruple when the size is increased. A more useful measure would be energy. It can be seen that from the 8x8 array to the 16x16 array, the energy used increases by almost three fold. This is because with the increase in array size, the run time is increased to compensate for the extra columns. Also, the simulator needed more time to calculate the results in the larger arrays.

The edge image in both the schematic and extracted results reveals all the edges in the image. The image displayed in Figure 5.20 is larger than 16x16, thus the test case was taken from the right of the image. Compared to the edge images, it can be seen that all edges, especially the two cutting diagonally across, are detected.

5.4.7 Pixel Overall Specifications

Table 5.5 summarizes the pixels overall specifications, where an increased fill factor is achieved. The method of gray-scale erosion using bitwise decomposition's functionality is verified and the goal of a compact edge detection pixel array is reached. The arrays power consumption is minimal, making the design scalable without overwhelming power usage.

The pixels dynamic range is well within the acceptable range.

Table 5.5: Summary of pixel specifications

Technology	0.13 μm
Supply Voltage	1 V
Pixel Size	17x17 μm
Photodiode Size	8x8 μm
Transistors per Pixel	107
Fill Factor	22%
Dynamic Range	46 dB
Photo-detector Type	Photodiode
Power Dissipation (16x16)	0.38 mW
Read out time (16x16)	1.559 μs
ADC Architecture	Per-Pixel Single Slope
ADC Range	1V
ADC Resolution	8-bits
ADC Conversion Time	40 μsec

5.4.8 Conclusion

The design of the pixel involves components with minimal number of transistors. This gives the pixel an overall transistor count of 107. Control lines were also kept to a minimum. The design uses many PTL components, minimizing the design area.

Layout of the pixel has achieved the desired increase in fill factor compared to other designs. The pixel size does not compromise resolution, yet it achieves it goals.

The design reaches its goals of a portable imaging device aimed at visual prosthetics. Table 5.6 compares the current design with a summary of others as mentioned in the literature review. The designs that it is compared to are those with on-pixel ADC and any processing power.

Table 5.6: Comparison of the proposed design with those from the literature, summarized in Table 1.2

	Other Designs	**Proposed Design**
Fill Factor	$\leq$10%	22%
Signal Control	SIMD and Integer Pipelined for controller	Simple Control Unit. FSM
Signal Processing	Microprocessor	Simple unit for edge detection. (subtractor, comparator)
Erosion Type	Binary	Gray-scale
Pixel Size	Large (67.4x67.4 μm^2)	Reasonable (17x17 μm^2)

Chapter 6

Conclusions and Future Work

Edge images are the most useful to blind patients. Edge detection algorithms are computationally intensive and cannot be implemented on pixel. Using mathematical morphology simplifies the process but current on-pixel implementations are limited to binary images. This means the loss of detail in the image.

Many designs incorporate on pixel processing power. Most have partial image processing capabilities and binary mathematical morphology only. These designs have achieved low fill factors of 10% or less, with large pixel sizes.

The design in this work was inspired by binary mathematical morphology. By using gray-scale bitwise decomposition into binary morphology, erosion can be implemented using AND gates. Gray-scale mathematical morphology can be implemented on pixel. This means a larger fill factor, and a smaller processor and control unit. The portability of the imager and its processor make it ideal for visual prosthetics such as the epiretinal implant or the VCI.

6.1 Design Summary

The design proposed in this thesis consisted of on-pixel Gray-scale Mathematical Erosion. The control unit would gather the image values and the erosion output. It would then carry out edge detection using two erosions for a noise robust result.

The design is based on gray-scale bitwise decomposition into binary erosion aimed at on-pixel implementation and edge detection. This approach is tested and had been proven to work under several noisy conditions.

Schematic and extracted results have shown that the proposed method of gray-scale erosion using bitwise decomposition is feasible in hardware.

6.2 Thesis Contributions

This thesis has presented a new method of implementing gray-scale erosion. Aimed at on-pixel implementation, the proposed method allows for a more portable imaging device. In summary, the thesis has:

1. Proposed a new gray-scale erosion method. The new erosion technique is used in an edge detection algorithm to yield results comparable with classical algorithms. This new method of gray-scale erosion allows for on-pixel implementation. Combined with morphological edge detection, it creates a portable edge detection imager.

2. The fill factor of the pixel is increased. Pixels employing any extra processing capabilities achieve a fill factor of less than or equal to 10%. The fill factor achieved in this design is 22% and the dynamic range as 46 dB.

3. Portability is increased in this design, with the elimination of the need for large microprocessors. The pixel array needs a simple control unit to generate the signals and finish of the edge detection.

4. Aimed at visual prosthetics, the design delivers an sketch that is realistic and clear. The image is made of dots closely representing the phosphenes that blind patients with visual cortical implants experience. The design is not limited to visual prosthetics, but can be used where portable edge detection is needed.

6.3 Future Work

Future work on this topic would include the completion of the imaging device by implementing the control units and the sense amps. The fabricated control chip should include the remainder of the edge detection process. Amorphous silicon photodiodes can be used to increase the light collecting efficiency. The array should be fabricated and tested with a lens installed. Performance can also be improved by optimizing the transistor sizes. It would be interesting to implement the design in the newer and smaller technologies, this could achieve an even better fill factor. The proposed algorithm should also be tested

further to provide more details on its operation.

A more useful contribution to the imager would be infrared sensing. Photodiodes can be manufactured to detect infrared light. They have been in use in subretinal implants by Optobioincs [30]. Instead of having one photodiode, each pixel would have two, one being the regular type, the other infrared sensitive. If the infrared diode is active, then that edge could be shown as red or any other colour. It was demonstrated that by varying the threshold current to the electrodes in the visual cortex, patients were able to see different colours [29]. Thus, wherever an infrared object's edge is detected, the implant can present it to the patient as a coloured edge. This would be useful to the patient in detecting inanimate objects, such as a pen versus a person, and in avoiding dangerous objects, such as a heating element that is on.

The imaging device is not limited to only visual cortical implants. It can be used in any application requiring portable edge detection. Such an application includes fingerprint detection, biometrics and medical imaging.

Appendix A

Matlab Code Used in Testing the Algorithm and Array Output

A.1 Matlab Commands used to Compare Algorithms

Read an image:

Img=imread('image path and name'); Can treat Img like any other 2D array.

To find the Image size:

Rc=size(Img); Rc is a 1D array with the row then column size.

To Display an Image:

imshow(Img)

Convert an image from colour to greyscale:

ImgGS=rgb2gray(Img);

Convert an image from greyscale to binary:

ImgB=dither(ImgGS);

Edge detect an image using the Canny algorithm:

ImgCanny=edge(ImgGS,'canny');

Edge detect and image using Sobel algorithm:

ImgSobel=edge(ImgGS,'sobel');

To Crop an Image:

First load and show the image. Imshow(Img)

Then: ImgCrop=imcrop; Move the cursor and select the area to crop.

Adding noise to an Image:

Gaussian: ImgNoise=imnoise(Img,'gaussian',0,0.001);

Salt and Pepper: ImgNoise=imnoise(Img,'salt & pepper'); Speckle:

A.2 Matlab Code Written to Implement the Proposed Algorithm

```
function [edged]= mydetect(img)
    %defining the vairbles used
    rc=size(img);
    %getting the size of the image
    row=rc(1,1);
    col=rc(1,2);
    %defining the neighbour pixel coordinates, clouckwise, p1 starts at the %right
    r=row
    c=column
    p1r=0;p2r=0;p3r=0;p4r=0;p5r=0;p6r=0;p7r=0;p8r=0;
    p1c=0;p2c=0;p3c=0;p4c=0;p5c=0;p6c=0;p7c=0;p8c=0;
    %erosion1 is from domain 1, horizontal and vertical se
    erosion1=[row,col];
    %erosion2 is from domain 2, diagonal se
    erosion2=[row,col];
    %result1 is the intermediate result for edge detection
    result1=[row,col];
    %this is the resulting edge image
    edged=[row,col];
    %loop through the image, column then rows
    for i=1:row
    for j=1:col
    %the coordinates of the neigbors
    p1r=i; p1c=j+1; p2r=i-1; p2c=j+1; p3r=i-1; p3c=j;
    p4r=i-1; p4c=j-1; p5r=i; p5c=j-1; p6r=i+1; p6c=j-1;
    p7r=i+1; p7c=j; p8r=i+1; p8c=j+1;
    %the switch statement takes care of the edges and corners
    %the cases are rearranged such that corners are first.
    switch col>1
    %case 2, pixel on the upper right corner
    case ((p1c>col) & (p2c>col) & (p2r<1) & (p3r<1) & (p4r<1) & (p8c>col))
```

```
erosion1(i,j)=bitand(uint8(img(p5r,p5c)),uint8(img(p7r,p7c)));
erosion2(i,j)=img(p6r,p6c);
%case 4, pixel on the upper left corner
case ((p2r<1) & (p3r<1) & (p4r<1) & (p4c<1) & (p5c<1) & (p6c<1))
erosion1(i,j)=bitand(uint8(img(p1r,p1c)),uint8(img(p7r,p7c)));
erosion2(i,j)=img(p8r,p8c);
%case 6, pixel on the lower left corner
case ((p6r>row) & (p7r>row) & (p5c<1) & (p6c<1) & (p4c<1) & (p8r> row))
erosion1(i,j)=bitand(uint8(img(p1r,p1c)),uint8(img(p3r,p3c)));
erosion2(i,j)=img(p2r,p2c);
%case 8, pixel on the lower right corner
case ((p6r>row) & (p7r>row) & (p1c>col) & (p8c>col) & (p2c>col) & (p8r>row))
erosion1(i,j)=bitand(uint8(img(p5r,p5c)),uint8(img(p3r,p3c)));
erosion2(i,j)=img(p4r,p4c);
%case 1, pixel on the right edge
case (p1c>col)&(p2c>col)&(p8c>col)
temp=bitand(uint8(img(p3r,p3c)),uint8(img(p5r,p5c)));
erosion1(i,j)=bitand(uint8(temp),uint8(img(p7r,p7c)));
erosion2(i,j)=bitand(uint8(img(p4r,p4c)),uint8(img(p6r,p6c)));
%case 3, pixel on the upper edge
case ((p2r<1) & (p3r<1) & (p4r<1))
temp=bitand(uint8(img(p1r,p1c)),uint8(img(p5r,p5c)));
erosion1(i,j)=bitand(uint8(temp),uint8(img(p7r,p7c)));
erosion2(i,j)=bitand(uint8(img(p6r,p6c)),uint8(img(p6r,p6c)));
%case 5, pixel on the left edge
case ((p4c<1) & (p5c<1) & (p6c<1))
temp=bitand(uint8(img(p1r,p1c)),uint8(img(p3r,p3c)));
erosion1(i,j)=bitand(uint8(temp),uint8(img(p7r,p7c)));
erosion2(i,j)=bitand(uint8(img(p2r,p2c)),uint8(img(p8r,p8c)));
%case 7, pixel on the lower edge
case ((p6r>row) & (p7r>row) & (p8r>row))
temp=bitand(uint8(img(p3r,p3c)),uint8(img(p5r,p5c)));
erosion1(i,j)=bitand(uint8(temp),uint8(img(p1r,p1c)));
```

```
erosion2(i,j)=bitand(uint8(img(p4r,p4c)),uint8(img(p2r,p2c)));
% default case is when the pixel is in the middle
otherwise
temp=bitand(uint8(img(p3r,p3c)),uint8(img(p5r,p5c)));
tempc=bitand(uint8(img(p7r,p7c)),uint8(img(p1r,p1c)));
erosion1(i,j)=bitand(uint8(temp),uint8(tempc));
temp=bitand(uint8(img(p6r,p6c)),uint8(img(p8r,p8c)));
tempc=bitand(uint8(img(p4r,p4c)),uint8(img(p2r,p2c)));
erosion2(i,j)=bitand(uint8(temp),uint8(tempc));
end
end
end
%now to compute the intermediate result, according to the equatoin
%yet another switch statment is need to handle the corners and edges
for i=1:row
for j=1:col
%the coordinates of the neigbors
p1r=i; p1c=j+1; p2r=i-1; p2c=j+1; p3r=i-1; p3c=j;
p4r=i-1; p4c=j-1; p5r=i; p5c=j-1; p6r=i+1; p6c=j-1;
p7r=i+1; p7c=j; p8r=i+1; p8c=j+1;
%the switch statement takes care of the edges and corners
switch col>1
%case 2, pixel on the upper right corner
case ((p1c>col) & (p2c>col) & (p2r<1) & (p3r<1) & (p4r<1) & (p8c>col))
result1(i,j)=abs(img(i,j)-erosion1(p6r,p6c));
%case 4, pixel on the upper left corner
case ((p2r<1) & (p3r<1) & (p4r<1) & (p4c<1) & (p5c<1) & (p6c<1))
result1(i,j)=abs(img(i,j)-erosion1(p8r,p8c));
%case 6, pixel on the lower left corner
case ((p6r>row) & (p7r>row) & (p5c<1) & (p6c<1) & (p4c<1) & (p8r>row))
result1(i,j)=abs(img(i,j)-erosion1(p2r,p2c));
%case 8, pixel on the lower right corner
case ((p6r>row) & (p7r>row) & (p1c>col) & (p8c>col) & (p2c>col) & (p8r>row))
```

```
result1(i,j)=abs(img(i,j)-erosion1(p4r,p4c));
%case 1, pixel on the right edge
case (p1c>col)&(p2c>col)&(p8c>col)
result1(i,j)=max(abs(img(i,j)-erosion1(p4r,p4c)),abs(img(i,j)-erosion1(p6r,p6c)));
%case 3, pixel on the upper edge case ((p2r<1) & (p3r<1) & (p4r<1))
result1(i,j)=max(abs(img(i,j)-erosion1(p8r,p8c)),abs(img(i,j)-erosion1(p6r,p6c)));
%case 5, pixel on the left edge
case ((p4c<1) & (p5c<1) & (p6c<1))
result1(i,j)=max(abs(img(i,j)-erosion1(p8r,p8c)),abs(img(i,j)-erosion1(p2r,p2c)));
%case 7, pixel on the lower edge
case ((p6r>row) & (p7r>row) & (p8r> row))
result1(i,j)=max(abs(img(i,j)-erosion1(p4r,p4c)),abs(img(i,j)-erosion1(p2r,p2c)));
% default case is when the pixel is in the middle
otherwise
result1(i,j)=max(abs((img(i,j)-erosion1(p4r,p4c))-(img(i,j)-erosion1(p4r,p4c))),abs((img(i,j)-
erosion1(p2r,p2c))-(img(i,j)-erosion1(p6r,p6c))));
end
end
end
%now to compute the end result
for i=1:row
for j=1:col
%the coordinates of the neigbors
p1r=i; p1c=j+1; p2r=i-1; p2c=j+1; p3r=i-1; p3c=j;
p4r=i-1; p4c=j-1; p5r=i; p5c=j-1; p6r=i+1; p6c=j-1;
p7r=i+1; p7c=j; p8r=i+1; p8c=j+1;
temp=min(img(i,j)-erosion1(i,j),img(i,j)-erosion2(i,j));
edged(i,j)=min(temp,result1(i,j));
%thresholding the image to clear it up. lower the value for
%darker images
if (edged(i,j)<20)
edged(i,j)=0;
%uncomment the else below for binary images
```

```
else
edged(i,j)=1;
end end end
end
```

A.3 Matlab Code Written to Test and Compare the Array Output

```
function [edge,sim,img,temp]= morph() row=16; col=16;
    edge=[row,col];
    sim=[row,col];
    img=[row,col];
    temp=0;
    tempb=0;
    %defining the neighbour pixel coordinates, clouckwise, p1 starts at the %right
    r=row
    c=column
    p1r=0;p2r=0;p3r=0;p4r=0;p5r=0;p6r=0;p7r=0;p8r=0;
    p1c=0;p2c=0;p3c=0;p4c=0;p5c=0;p6c=0;p7c=0;p8c=0;
    %erosion1 is from domain 1, horizontal and vertical se
    erosion1=[row,col];
    D1=[row,col];
    %erosion2 is from domain 2, diagonal se
    erosion2=[row,col];
    D2=[row,col];
    %result1 is the intermediate result for edge detection
    result1=[row,col];
    result1b=[row,col];
    fid=fopen('img2_L.txt','r');
    fidD1=fopen('test2D1_L.txt','r');
    fidD2=fopen('test2D2_L.txt','r');
    fidr2=fopen('res2.txt','w');
    fidr2s=fopen('res2sim.txt','w');
```

```
fidr2D1=fopen('res2D1.txt','w');
fidr2D2=fopen('res2D2.txt','w');
%read in the files
for i=1:row
for j=1:col
img(i,j)=fscanf(fid,'%x',1);
D1(i,j)=fscanf(fidD1,'%x',1);
D2(i,j)=fscanf(fidD2,'%x',1);
end end
%find the two domains and write to the file
for i=1:row
for j=1:col
%the coordinates of the neigbors
p1r=i; p1c=j+1; p2r=i-1; p2c=j+1; p3r=i-1; p3c=j;
p4r=i-1; p4c=j-1; p5r=i; p5c=j-1; p6r=i+1; p6c=j-1;
p7r=i+1; p7c=j; p8r=i+1; p8c=j+1;
%the switch statement takes care of the edges and corners
%the cases are rearranged such that corners are first.
switch col>1
%case 2, pixel on the upper right corner
case ((p1c>col) & (p2c>col) & (p2r<1) & (p3r<1) & (p4r<1) & (p8c>col))
erosion1(i,j)=bitand(uint8(img(p5r,p5c)),uint8(img(p7r,p7c)));
erosion2(i,j)=img(p6r,p6c);
%case 4, pixel on the upper left corner
case ((p2r<1) & (p3r<1) & (p4r<1) & (p4c<1) & (p5c<1) & (p6c<1))
erosion1(i,j)=bitand(uint8(img(p1r,p1c)),uint8(img(p7r,p7c)));
erosion2(i,j)=img(p8r,p8c);
%case 6, pixel on the lower left corner
case ((p6r>row) & (p7r>row) & (p5c<1) & (p6c<1) & (p4c<1) & (p8r>row))
erosion1(i,j)=bitand(uint8(img(p1r,p1c)),uint8(img(p3r,p3c)));
erosion2(i,j)=img(p2r,p2c);
%case 8, pixel on the lower right corner
case ((p6r>row) & (p7r>row) & (p1c>col) & (p8c>col) & (p2c>col) & (p8r>row))
```

```
erosion1(i,j)=bitand(uint8(img(p5r,p5c)),uint8(img(p3r,p3c)));
erosion2(i,j)=img(p4r,p4c);
%case 1, pixel on the right edge
case (p1c>col)&(p2c>col)&(p8c>col)
temp=bitand(uint8(img(p3r,p3c)),uint8(img(p5r,p5c)));
erosion1(i,j)=bitand(uint8(temp),uint8(img(p7r,p7c)));
erosion2(i,j)=bitand(uint8(img(p4r,p4c)),uint8(img(p6r,p6c)));
%case 3, pixel on the upper edge
case ((p2r<1) & (p3r<1) & (p4r<1))
temp=bitand(uint8(img(p1r,p1c)),uint8(img(p5r,p5c)));
erosion1(i,j)=bitand(uint8(temp),uint8(img(p7r,p7c)));
erosion2(i,j)=bitand(uint8(img(p6r,p6c)),uint8(img(p8r,p8c)));
%case 5, pixel on the left edge
case ((p4c<1) & (p5c<1) & (p6c<1))
temp=bitand(uint8(img(p1r,p1c)),uint8(img(p3r,p3c)));
erosion1(i,j)=bitand(uint8(temp),uint8(img(p7r,p7c)));
erosion2(i,j)=bitand(uint8(img(p2r,p2c)),uint8(img(p8r,p8c)));
%case 7, pixel on the lower edge
case ((p6r>row) & (p7r>row) & (p8r>row))
temp=bitand(uint8(img(p3r,p3c)),uint8(img(p5r,p5c)));
erosion1(i,j)=bitand(uint8(temp),uint8(img(p1r,p1c)));
erosion2(i,j)=bitand(uint8(img(p4r,p4c)),uint8(img(p2r,p2c)));
% default case is when the pixel is in the middle
otherwise
temp=bitand(uint8(img(p3r,p3c)),uint8(img(p5r,p5c)));
tempc=bitand(uint8(img(p7r,p7c)),uint8(img(p1r,p1c)));
erosion1(i,j)=bitand(uint8(temp),uint8(tempc));
temp=bitand(uint8(img(p6r,p6c)),uint8(img(p8r,p8c)));
tempc=bitand(uint8(img(p4r,p4c)),uint8(img(p2r,p2c)));
erosion2(i,j)=bitand(uint8(temp),uint8(tempc));
end
fprintf(fidr2D1,'%x ',erosion1(i,j));
fprintf(fidr2D2,'%x ',erosion2(i,j));
```

```
end
end
%now calculate the intermedite results for
i=1:row
for j=1:col
p1r=i; p1c=j+1; p2r=i-1; p2c=j+1; p3r=i-1; p3c=j;
p4r=i-1; p4c=j-1; p5r=i; p5c=j-1; p6r=i+1; p6c=j-1;
p7r=i+1; p7c=j; p8r=i+1; p8c=j+1;
%the switch statement takes care of the edges and corners
switch col>1
%case 2, pixel on the upper right corner
case ((p1c>col) & (p2c>col) & (p2r<1) & (p3r<1) & (p4r<1) & (p8c>col))
result1(i,j)=abs(img(i,j)-erosion1(p6r,p6c));
result1b(i,j)=abs(img(i,j)-D1(p6r,p6c));
%case 4, pixel on the upper left corner
case ((p2r<1) & (p3r<1) & (p4r<1) & (p4c<1) & (p5c<1) & (p6c<1))
result1(i,j)=abs(img(i,j)-erosion1(p8r,p8c));
result1b(i,j)=abs(img(i,j)-D1(p8r,p8c));
%case 6, pixel on the lower left corner
case ((p6r>row) & (p7r>row) & (p5c<1) & (p6c<1) & (p4c<1) & (p8r>row))
result1(i,j)=abs(img(i,j)-erosion1(p2r,p2c));
result1b(i,j)=abs(img(i,j)-D1(p2r,p2c));
%case 8, pixel on the lower right corner
case ((p6r>row) & (p7r>row) & (p1c>col) & (p8c>col) & (p2c>col) & (p8r>row))
result1(i,j)=abs(img(i,j)-erosion1(p4r,p4c));
result1b(i,j)=abs(img(i,j)-D1(p4r,p4c));
%case 1, pixel on the right edge
case (p1c>col)&(p2c>col)&(p8c>col)
result1(i,j)=max(abs(img(i,j)-erosion1(p4r,p4c)),abs(img(i,j)-erosion1(p6r,p6c)));
result1b(i,j)=max(abs(img(i,j)-D1(p4r,p4c)),abs(img(i,j)-D1(p6r,p6c)));
%case 3, pixel on the upper edge
case ((p2r<1) & (p3r<1) & (p4r<1))
result1(i,j)=max(abs(img(i,j)-erosion1(p8r,p8c)),abs(img(i,j)-erosion1(p6r,p6c)));
```

```
result1b(i,j)=max(abs(img(i,j)-D1(p8r,p8c)),abs(img(i,j)-D1(p6r,p6c)));
%case 5, pixel on the left edge
case ((p4c<1) & (p5c<1) & (p6c<1))
result1(i,j)=max(abs(img(i,j)-erosion1(p8r,p8c)),abs(img(i,j)-erosion1(p2r,p2c)));
result1b(i,j)=max(abs(img(i,j)-D1(p8r,p8c)),abs(img(i,j)-D1(p2r,p2c)));
%case 7, pixel on the lower edge
case ((p6r>row) & (p7r>row) & (p8r>row))
result1(i,j)=max(abs(img(i,j)-erosion1(p4r,p4c)),abs(img(i,j)-erosion1(p2r,p2c)));
result1b(i,j)=max(abs(img(i,j)-D1(p4r,p4c)),abs(img(i,j)-D1(p2r,p2c)));
% default case is when the pixel is in the middle
otherwise
result1(i,j)=max(abs((img(i,j)-erosion1(p4r,p4c))-(img(i,j)-erosion1(p4r,p4c))),
abs((img(i,j)-erosion1(p2r,p2c))-(img(i,j)-erosion1(p6r,p6c))));
result1b(i,j)=max(abs((img(i,j)-D1(p4r,p4c))-(img(i,j)-D1(p4r,p4c))),
abs((img(i,j)-D1(p2r,p2c))-(img(i,j)-D1(p6r,p6c))));
end
end
end
for i=1:row
for j=1:col
%the coordinates of the neigbors
temp=min((img(i,j)-erosion1(i,j)),(img(i,j)-erosion2(i,j)));
edge(i,j)=min(temp,result1(i,j));
tempb=min(img(i,j)-D1(i,j),img(i,j)-D2(i,j));
sim(i,j)=min(tempb,result1b(i,j));
if (edge(i,j)<20)
edge(i,j)=0;
else
edge(i,j)=1;
end
if (sim(i,j)<20)
sim(i,j)=0;
%uncomment the else below for binary images
```

```
else
sim(i,j)=1;
end
fprintf(fidr2,'%x ',edge(i,j));
fprintf(fidr2s,'%x ',img(i,j));
img2(i,j)=abs(edge(i,j)-sim(i,j));
end end
fclose('all');
%function end
end
```

Appendix B

Verilog Code Used in Generating the Control Signals to the Arrays

B.1 Verilog Code written for the Control of a Single Pixel

```
//Verilog-AMS HDL for "erosion_DPS", "pixel_sig_gen" "verilogams"
    //anything that ends with i is an internal value
    'include "constants.vams" 'include "disciplines.vams"
    module pixel_sig_gen (
    //Pixel signals
    Vgp,Vgn,
    //Tristate buffers
    W_TS,NW_TS,
    //Bias ckt
    PE,NPE,
    //Latchs
    AEN,NAEN,NEN,NNEN,
    //MUXS
    S0,S1,S2,
    //WRITE/READS
    R,NR,NNR,NRB,NNRB,
    //DOMAIN 1
    N,W,S,E,
```

```
//DOMAIN 2
NW,NE,SW,SE,
//OUTPUTS
bit, D1, D2,
//counter
en, rst );
//seting up the I/O
//      Outputs------
output
//Pixel signals
Vgp,Vgn,
//Tristate buffers
W_TS,NW_TS,
//Bias ckt
PE,NPE,
//Latchs
AEN,NAEN,NEN,NNEN,
//MUXS
S0,S1,S2,
//WRITE/READS
R,NR,NNR,NRB,NNRB,
//DOMAIN 1
N,W,S,E,
//DOMAIN 2
NW,NE,SW,SE;
//      inputs------
input bit, D1,D2;
//-------IO
//DATA BUS
output en, rst;
//Latchs
reg AEN,NAEN,NEN,NNEN,
//Tristate buffers
```

```
W_TS,NW_TS,
//Bias ckt
PE,NPE,
//MUXS
S0,S1,S2,
//WRITE/READS
R,NR,NNR, NRB,NNRB,
//DOMAIN 1
N,W,S,E,
//DOMAIN 2
NW,NE,SW,SE;
//OUTPUTS
//bit,
//DATA BUS
reg //Pixel signals
Vgp,Vgn,en, rst;
//-    -internal variables
//these reg hold the values for the pixels
//THIS HOLDS THE CURRENT PIXEL VALUE
reg [7:0]pixel;
reg [7:0]Ni; reg [7:0]NWi;reg [7:0]NEi; reg [7:0]Wi; reg [7:0]Ei;
reg [7:0]Si;
reg [7:0]SWi; reg [7:0]SEi;
reg [7:0]D1i;
reg [7:0]D2i; //these hold the output of the erosion
integer i;
//-    -code starts here
initial begin
$shm_open("waves"); $shm_probe("AS");
//place values into the pixels
pixel=0; D1i=0; D2i=0;
Ni =20; NEi=255; NWi=255; Wi=60;SWi=0;
Si=100; SEi=0; Ei=55;
```

```
//RESET STATE
AEN=0;NAEN=1;NEN=0;NNEN=1;
NPE=1; PE=0;
NW_TS=1; W_TS=0;
//MUXS
S0=0;S1=0;S2=0;
//WRITE/READS
R=0;NR=0;NNR=1;
NRB=0;NNRB=1;
//DOMAIN 1
N=0;W=0;S=0;E=0;
//DOMAIN 2
NW=0;NE=0;SW=0;SE=0;
Vgp=0; Vgn=0;
//DATA BUS
rst = 1; en = 0;
#1 Vgp=1;
rst=0;
NPE=0; PE=1;
//Aquire the image and convert to digital
#50;
rst = 0;
Vgp=0; Vgn=0;
NW_TS=0; W_TS=1;
NPE=0; PE=1;
//    start of cycle 2
#140;
en=1;
Vgp=0; Vgn=0;
NPE=0; PE=1;
NW_TS=0; W_TS=1;
//start the erosion
#39805
```

```
en=0;
#35
en=0;
Vgp=0;
rst=0;
NPE=1; PE=0;
NW_TS=1; W_TS=0;
R=1;
NEN=1; NNEN=0;
AEN=1;NAEN=0;
NR=1;NNR=0;
NRB=1; NNRB=0;
for (i=0; i<9; i=i+1)
begin
S0=i[0];S1=i[1];S2=i[2]; //------ Cycle 2 ------
//latch this pixels value
#5
N=Ni[i];NW=NWi[i];NE=NEi[i];E=Ei[i];W=Wi[i];SW=SWi[i];SE=SEi[i];S=Si[i];
//------ Cycle 3 ------
//start the erosion by and
//place values into the neighbors
D1i[i]=D1;
D2i[i]=D2;
//------ Cycle 4 ------
//LATCH THE VALUE
pixel[i]=bit;
end
end
endmodule
```

B.2 Verilog Code written for the Control of a Single Processing Element

```
//Verilog-AMS HDL for "erosion_DPS", "PE2_sig_gen" "verilogams"
  //This module tests the PE of the DPS
  // It tests only one PE and without the pixels (PD)
  //The pixel values are generated here
  // Duha Jabakhanji
  'include "constants.vams" 'include "disciplines.vams"
  module PE2_sig_gen (
  //Latchs
  AEN,NAEN,NEN,NNEN,
  //MUXS
  S0,S1,S2,
  //WRITE/READS
  R,NR,NNR,Wr,NRB,NNRB,
  //DOMAIN 1
  N,W,S,E,
  //DOMAIN 2
  NW,NE,SW,SE,
  //OUTPUTS
  bit, D1, D2,
  //DATA BUS
  IO0,IO1,IO2,IO3,IO4,IO5,IO6,IO7,
  );
  //seting up the I/O
  //------ Outputs
  output //Latchs
  AEN,NAEN,NEN,NNEN,
  //MUXS
  S0,S1,S2,
  //WRITE/READS
  R,Wr,NR,NNR,NRB,NNRB,
  //DOMAIN 1
```

```
N,W,S,E,
//DOMAIN 2
NW,NE,SW,SE;
//-------inputs-------
input bit, D1,D2;
//-------IO-----------
//DATA BUS
output IO0,IO1,IO2,IO3,IO4,IO5,IO6,IO7;
//Latchs
reg AEN,NAEN,NEN,NNEN,
//MUXS
S0,S1,S2,
//WRITE/READS
R,Wr,NR,NNR,NRB,NNRB,
//DOMAIN 1
N,W,S,E,
//DOMAIN 2
NW,NE,SW,SE;
//OUTPUTS
wire bit;
//DATA BUS
reg IO0,IO1,IO2,IO3,IO4,IO5,IO6,IO7;
//-----  -internal variables----------
//these reg hold the values for the pixels
//THIS HOLDS THE CURRENT PIXEL VALUE
reg [7:0]pixel;
reg [7:0]Ni; reg [7:0]NWi;reg [7:0]NEi; reg [7:0]Wi; reg [7:0]Ei;
reg [7:0]Si;
reg [7:0]SWi; reg [7:0]SEi;
reg clk;
//these hold the output of the erosion
reg [7:0]D1i;
reg [7:0]D2i;
```

```
reg [7:0]pixeli;
integer i;
//      -code starts here
initial begin
$shm_open("waves");
$shm_probe("AS");
//place values into the pixels
pixel = 55; Ni =20; NEi=255; NWi=255; Wi=60;SWi=255;
Si=100; SEi=255; Ei=55;
//RESET STATE
AEN=0;NAEN=1;NEN=0;NNEN=1;
//MUXS
S0=0;S1=0;S2=0;
//WRITE/READS
R=0;Wr=0;NR=0;NNR=1; NRB=0; NNRB=1;
//DOMAIN 1
N=0;W=0;S=0;E=0;
//DOMAIN 2
NW=0;NE=0;SW=0;SE=0;
//DATA BUS
IO0=0;
IO1=0;IO2=0;IO3=0;IO4=0;IO5=0;IO6=0;IO7=0;
clk=0;
//          Cycle 1
//Place pixel value into DRAM
#1 IO0=pixel[0]; IO1=pixel[1];IO2=pixel[2];IO3=pixel[3];IO4=pixel[4];IO5=pixel[5];
IO6=pixel[6];IO7=pixel[7];
//  write enable
Wr=1;
#1
Wr=0;
AEN=1;NAEN=0;
IO0=1'b Z; IO1=1'b Z;IO2=1'b Z;IO3=1'b Z;IO4=1'b Z;IO5=1'b Z;
```

```
IO6=1'b Z;IO7=1'b Z;
#40000
for (i=0; i<9; i=i+1)
begin
//          Cycle 2
//latch this pixels value
#5
R=1;
NEN=1; NNEN=0;
N=Ni[i];NW=NWi[i];NE=NEi[i];E=Ei[i];W=Wi[i];SW=SWi[i];SE=SEi[i];S=Si[i];
NR=1;NNR=0;
NRB=1; NNRB=0;
//          Cycle 3
//start the erosion by and
//place values into the neighbors
#1.5
S0=i[0];S1=i[1];S2=i[2];
D1i[i]=D1;
D2i[i]=D2;
NEN=0; NNEN=1;
//          Cycle 4
//LATCH THE VALUE
if (i¿0) begin
//for some reason, the verilog code reads in the value from the previous cycle
pixeli[i-1]=bit;
end
end
end
endmodule
```

B.3 Verilog Code written for the 4x4 Pixel array

B.3.1 Verilog Code Written to Load an Image into the 4x4 Array

```
//Verilog-AMS HDL for "Arrays", "image_gen4" "verilogams"
  //This code reads a 16X16 image array from a file and loads it into the pixel
  //Duha Jabakhanji
  //Jul4 2006
  //`include "constants.vams"
  //`include "disciplines.vams"
  module image_gen4 ( output
  //Row 0, BITS 0 TO 7
  R0B0,R0B1,R0B2,R0B3,R0B4,R0B5,R0B6,R0B7,
  //Row 1, BITS 0 TO 7
  R1B0,R1B1,R1B2,R1B3,R1B4,R1B5,R1B6,R1B7,
  //Row 2, BITS 0 TO 7
  R2B0,R2B1,R2B2,R2B3,R2B4,R2B5,R2B6,R2B7,
  //Row 3, BITS 0 TO 7
  R3B0,R3B1,R3B2,R3B3,R3B4,R3B5,R3B6,R3B7,
  //WRITE SIGNALS
  WC0,WC1,WC2,WC3 );
  reg
  //Row 0, BITS 0 TO 7
  R0B0,R0B1,R0B2,R0B3,R0B4,R0B5,R0B6,R0B7,
  //Row 1, BITS 0 TO 7
  R1B0,R1B1,R1B2,R1B3,R1B4,R1B5,R1B6,R1B7,
  //Row 2, BITS 0 TO 7
  R2B0,R2B1,R2B2,R2B3,R2B4,R2B5,R2B6,R2B7,
  //Row 3, BITS 0 TO 7
  R3B0,R3B1,R3B2,R3B3,R3B4,R3B5,R3B6,R3B7,
  WC0,WC1,WC2,WC3;
  integer file, i; integer p;
  //memory to hold the image values
  reg [7:0]image[0:255];
```

```
initial begin
//$shm_open("waves");
//$shm_probe("AS");
//initialize the pointers
//position indicates the next spot
//WRITE SIGNALS
WC0=0;WC1=0;WC2=0;WC3=0;
#1 //read the file into the memory
$readmemh("/home/duha/research/test2h.txt",image);
//loop is for the column
for (i=0; i<16; i=i+1) begin
#1
p=4;
//Row 0, BITS 0 TO 7
R0B0=image[i][0];R0B1=image[i][1];R0B2=image[i][2];R0B3=image[i][3];
R0B4=image[i][4];R0B5=image[i][5];R0B6=image[i][6];R0B7=image[i][7];
//Row 1, BITS 0 TO 7
R1B0=image[i+p][0];R1B1=image[i+p][1];R1B2=image[i+p][2];R1B3=image[i+p][3];
R1B4=image[i+p][4];R1B5=image[i+p][5];R1B6=image[i+p][6];R1B7=image[i+p][7];
//Row 2, BITS 0 TO 7
p=p+4;
R2B0=image[i+p][0];R2B1=image[i+p][1];R2B2=image[i+p][2];R2B3=image[i+p][3];
R2B4=image[i+p][4];R2B5=image[i+p][5];R2B6=image[i+p][6];R2B7=image[i+p][7];
//Row 3, BITS 0 TO 7
p=p+4;
R3B0=image[i+p][0];R3B1=image[i+p][1];R3B2=image[i+p][2];R3B3=image[i+p][3];
R3B4=image[i+p][4];R3B5=image[i+p][5];R3B6=image[i+p][6];R3B7=image[i+p][7];
//assert the write signals
if(i==0)begin WC0=1; end if(i==1) begin WC1=1; end if(i==2) begin
WC2=1; end if(i==3) begin WC3=1; end
#1
//deassert the write signals
WC0=0;WC1=0;WC2=0;WC3=0;
```

```
    end
    //once done, put the signals in tristate mode
    //Row 0, BITS 0 TO 7
    R0B0=1'b Z;R0B1=1'b Z ;R0B2=1'b Z ;R0B3=1'b Z ;R0B4=1'b Z ;R0B5=1'b Z
;R0B6=1'b Z ;R0B7=1'b Z ;
    //Row 1, BITS 0 TO 7
    R1B0=1'b Z;R1B1=1'b Z;R1B2=1'b Z ;R1B3=1'b Z ;R1B4=1'b Z ;R1B5=1'b Z ;R1B6=1'b
Z ;R1B7=1'b Z ;
    //Row 2, BITS 0 TO 7
    R2B0=1'b Z;R2B1=1'b Z;R2B2=1'b Z ;R2B3=1'b Z ;R2B4=1'b Z ;R2B5=1'b Z ;R2B6=1'b
Z ;R2B7=1'b Z ;
    //Row 3, BITS 0 TO 7
    R3B0=1'b Z;R3B1=1'b Z;R3B2=1'b Z ;R3B3=1'b Z ;R3B4=1'b Z ;R3B5=1'b Z ;R3B6=1'b
Z ;R3B7=1'b Z ;
    end
    endmodule
```

B.3.2 Verilog Code Written to Control the 4x4 Array and Read the Image and Erosion Results

```
//Verilog-AMS HDL for "Arrays", "CU_PE4" "verilogams"
    // This is the control unit for the 16X16 array with only the PE
    //Functions include:
    // Control signals on the array
    // Get data from the array and write them to a file
    //Finish off the edge detection
    //July 4 2006
    //`include "constants.vams"
    //`include "disciplines.vams"
    module CU_PE4 (
    // AEN and NEN as a bus
    AEN, NEN,
    //Read also as a bus
    Read,
```

```
// Tristate control as a bus
NR, NRB,
//The select value in a small bus
S,
//outputs of the array all as buses
//The two domains, D1 and D2
D1, D2, Img );
//declare the outputs
output [3:0]AEN,NEN,Read,NR, NRB; output [2:0] S;
//declare the inputs
input [3:0] D1,D2; input [15:0] Img;
//internal values
reg [3:0]AEN,NEN,Read,NR,NRB; reg [2:0] S;
//reg [15:0] D1,D2;
//counter values
integer i,j,k,d;
//file pointers
integer fid1, fid2,fid3;
//these hold the output of the array, bit by bit to collect the value
reg [7:0]BitD1 [0:255];
reg [7:0]BitD2 [0:255];
reg [7:0]Pic [0:255];
initial begin
$shm_open("waves");
$shm_probe("AS");
//initial state
AEN=0; NEN=0; S=0; Read=0; NR=0; NRB=0;
//open the files that will hold the outputs
fid1=$fopen ("/home/duha/research/Array4.txt");
fid2=$fopen ("/home/duha/research/Array4D1.txt");
fid3=$fopen ("/home/duha/research/Array4D2.txt");
#35
//This loop iterates to get the bits
```

```
for(i=0;i<8;i=i+1) begin
#0.25
S=i;
//This loops gets the values one column at a time
for(j=0;j<4;j=j+1) begin
//latch in the domain values
#0.25
//read mem, enable TS, and latch value into A
//read the values of the pixels around the current pixel
//this if is for the right edge
if(j>0)begin
AEN[j-1]=1;
NRB[j-1]=1;
Read[j-1]=1;
#1
AEN[j-1]=0;
Read[j-1]=0;
#0.5
d= d;
end
//this if is for the left edge
if(j<4) begin
AEN[j+1]=1;
NRB[j+1]=1;
Read[j+1]=1;
#1
AEN[j+1]=0;
Read[j+1]=0;
#0.5
d= d;
end
//read the current pixel column values
AEN[j]=1;
```

```
Read[j]=1;
NRB[j]=1;
NR[j]=1;
NEN[j]=1;
#2
//Read the values in
Pic[j][i]=Img[j];
k=4;
Pic[j+k][i]=Img[j+k];
k=k+4;
Pic[j+k][i]=Img[j+k];
k=k+4;
Pic[j+k][i]=Img[j+k];
BitD1[j][i]=D1[0];
k=4;
BitD1[j+k][i]=D1[1];
k=k+4; BitD1[j+k][i]=D1[2];
k=k+4;
BitD1[j+k][i]=D1[3];
//now for D2
BitD2[j][i]=D2[0];
k=4;
BitD2[j+k][i]=D2[1];
k=k+4;
BitD2[j+k][i]=D2[2];
k=k+4;
BitD2[j+k][i]=D2[3];
#0.5
//De-assert the signals
AEN[j]=0;
Read[j]=0;
NR[j]=0;
NEN[j]=0;
```

```
NRB[j]=0;
if(j>0)begin
NRB[j-1]=0;
end if(j<4) begin
NRB[j+1]=0;
end
//end of j (inner) loop
end
//end of i (outer) loop
end //these were working fine, better than fwrite
//$writememh("/home/duha/research/Array4D1.txt",BitD1);
//$writememh("/home/duha/research/Array4D2.txt",BitD2);
//$writememh("/home/duha/research/Array4.txt",Pic);
//write the values to the file from mem
for(i=0;i<16;i=i+1) begin
$fwrite(fid1,"%h ",Pic[i]);
$fwrite(fid2,"%h ",BitD1[i]);
$fwrite(fid3,"%h ",BitD2[i]); end
//close the files.
$fclose(fid1);
$fclose(fid2);
$fclose(fid3);
//end of initial block
end
endmodule
```

B.4 Verilog Code written for the 8x8 Pixel array

B.4.1 Verilog Code Written to Load an Image into the 8x8 Array

```
//Verilog-AMS HDL for "erosion_DPS", "image_gen8" "verilogams"
    //This code reads a 16X16 image array from a file and loads it into the pixel
    //Duha Jabakhanji
    //May 9 2006
```

```
//`include "/CMC/tools/cadence/IUS/tools.sun4v/spectre/etc/ahdl/constants.vams"
//`include "/CMC/tools/cadence/IUS/tools.sun4v/spectre/etc/ahdl/disciplines.vams"
module image_gen8 ( output
//Row 0, BITS 0 TO 7
R0B0,R0B1,R0B2,R0B3,R0B4,R0B5,R0B6,R0B7,
//Row 1, BITS 0 TO 7
R1B0,R1B1,R1B2,R1B3,R1B4,R1B5,R1B6,R1B7,
//Row 2, BITS 0 TO 7
R2B0,R2B1,R2B2,R2B3,R2B4,R2B5,R2B6,R2B7,
//Row 3, BITS 0 TO 7
R3B0,R3B1,R3B2,R3B3,R3B4,R3B5,R3B6,R3B7,
//Row 4, BITS 0 TO 7
R4B0,R4B1,R4B2,R4B3,R4B4,R4B5,R4B6,R4B7,
//Row 5, BITS 0 TO 7
R5B0,R5B1,R5B2,R5B3,R5B4,R5B5,R5B6,R5B7,
//Row 6, BITS 0 TO 7
R6B0,R6B1,R6B2,R6B3,R6B4,R6B5,R6B6,R6B7,
//Row 7, BITS 0 TO 7
R7B0,R7B1,R7B2,R7B3,R7B4,R7B5,R7B6,R7B7,
//WRITE SIGNALS
WC0,WC1,WC2,WC3,WC4,WC5,WC6,WC7 );
reg
//Row 0, BITS 0 TO 7
R0B0,R0B1,R0B2,R0B3,R0B4,R0B5,R0B6,R0B7,
//Row 1, BITS 0 TO 7
R1B0,R1B1,R1B2,R1B3,R1B4,R1B5,R1B6,R1B7,
//Row 2, BITS 0 TO 7
R2B0,R2B1,R2B2,R2B3,R2B4,R2B5,R2B6,R2B7,
//Row 3, BITS 0 TO 7
R3B0,R3B1,R3B2,R3B3,R3B4,R3B5,R3B6,R3B7,
//Row 4, BITS 0 TO 7
R4B0,R4B1,R4B2,R4B3,R4B4,R4B5,R4B6,R4B7,
//Row 5, BITS 0 TO 7
```

```
R5B0,R5B1,R5B2,R5B3,R5B4,R5B5,R5B6,R5B7,
//Row 6, BITS 0 TO 7
R6B0,R6B1,R6B2,R6B3,R6B4,R6B5,R6B6,R6B7,
//Row 7, BITS 0 TO 7
R7B0,R7B1,R7B2,R7B3,R7B4,R7B5,R7B6,R7B7,
WC0,WC1,WC2,WC3,WC4,WC5,WC6,WC7;
integer file, i;
integer p;
reg [7:0]image[0:255];
initial begin
//$shm_open("waves");
//$shm_probe("AS");
//initialize the pointers
//position indicates the next spot
//WRITE SIGNALS
WC0=0;WC1=0;WC2=0;WC3=0;WC4=0;WC5=0;WC6=0;WC7=0;
//open the file to be read
#1
//read the file into the memory
$readmemh("/home/duha/research/test2h.txt",image);
//loop is for the column
for (i=0; i<16; i=i+1) begin
#1
p=8;
//Row 0, BITS 0 TO 7
R0B0=image[i][0];R0B1=image[i][1];R0B2=image[i][2];R0B3=image[i][3];
R0B4=image[i][4];R0B5=image[i][5];R0B6=image[i][6];R0B7=image[i][7];
//Row 1, BITS 0 TO 7
R1B0=image[i+p][0];R1B1=image[i+p][1];R1B2=image[i+p][2];R1B3=image[i+p][3];
R1B4=image[i+p][4];R1B5=image[i+p][5];R1B6=image[i+p][6];R1B7=image[i+p][7];
//Row 2, BITS 0 TO 7
p=p+8;
R2B0=image[i+p][0];R2B1=image[i+p][1];R2B2=image[i+p][2];R2B3=image[i+p][3];
```

```
R2B4=image[i+p][4];R2B5=image[i+p][5];R2B6=image[i+p][6];R2B7=image[i+p][7];
//Row 3, BITS 0 TO 7
p=p+8;
R3B0=image[i+p][0];R3B1=image[i+p][1];R3B2=image[i+p][2];R3B3=image[i+p][3];
R3B4=image[i+p][4];R3B5=image[i+p][5];R3B6=image[i+p][6];R3B7=image[i+p][7];
//Row 4, BITS 0 TO 7
p=p+8;
R4B0=image[i+p][0];R4B1=image[i+p][1];R4B2=image[i+p][2];R4B3=image[i+p][3];
R4B4=image[i+p][4];R4B5=image[i+p][5];R4B6=image[i+p][6];R4B7=image[i+p][7];
//Row 5, BITS 0 TO 7
p=p+8;
R5B0=image[i+p][0];R5B1=image[i+p][1];R5B2=image[i+p][2];R5B3=image[i+p][3];
R5B4=image[i+p][4];R5B5=image[i+p][5];R5B6=image[i+p][6];R5B7=image[i+p][7];
//Row 6, BITS 0 TO 7
p=p+8;
R6B0=image[i+p][0];R6B1=image[i+p][1];R6B2=image[i+p][2];R6B3=image[i+p][3];
R6B4=image[i+p][4];R6B5=image[i+p][5];R6B6=image[i+p][6];R6B7=image[i+p][7];
//Row 7, BITS 0 TO 7
p=p+8;
R7B0=image[i+p][0];R7B1=image[i+p][1];R7B2=image[i+p][2];R7B3=image[i+p][3];
R7B4=image[i+p][4];R7B5=image[i+p][5];R7B6=image[i+p][6];R7B7=image[i+p][7];
//assert reads
if(i==0)begin WC0=1;
end
if(i==1) begin
WC1=1;
end
if(i==2) begin
WC2=1;
end
if(i==3) begin
WC3=1;
end
```

```
    if(i==4) begin
    WC4=1;
    end
    if(i==5) begin
    WC5=1;
    end
    if(i==6) begin
    WC6=1;
    end
    if(i==7) begin
    WC7=1;
    end
    #1
    //deassert reads
    WC0=0;WC1=0;WC2=0;WC3=0;WC4=0;WC5=0;WC6=0;WC7=0;
    end
    //once done, put the signals in tristate mode
    //Row 0, BITS 0 TO 7
    R0B0=1'b Z;R0B1=1'b Z ;R0B2=1'b Z ;R0B3=1'b Z ;R0B4=1'b Z ;R0B5=1'b Z
;R0B6=1'b Z ;R0B7=1'b Z ;
    //Row 1, BITS 0 TO 7
    R1B0=1'b Z;R1B1=1'b Z;R1B2=1'b Z ;R1B3=1'b Z ;R1B4=1'b Z ;R1B5=1'b Z ;R1B6=1'b
Z ;R1B7=1'b Z ;
    //Row 2, BITS 0 TO 7
    R2B0=1'b Z;R2B1=1'b Z;R2B2=1'b Z ;R2B3=1'b Z ;R2B4=1'b Z ;R2B5=1'b Z ;R2B6=1'b
Z ;R2B7=1'b Z ;
    //Row 3, BITS 0 TO 7
    R3B0=1'b Z;R3B1=1'b Z;R3B2=1'b Z ;R3B3=1'b Z ;R3B4=1'b Z ;R3B5=1'b Z ;R3B6=1'b
Z ;R3B7=1'b Z ;
    //Row 4, BITS 0 TO 7
    R4B0=1'b Z;R4B1=1'b Z;R4B2=1'b Z ;R4B3=1'b Z ;R4B4=1'b Z ;R4B5=1'b Z ;R4B6=1'b
Z ;R4B7=1'b Z ;
    //Row 5, BITS 0 TO 7
```

```
R5B0=1'b Z;R5B1=1'b Z;R5B2=1'b Z ;R5B3=1'b Z ;R5B4=1'b Z ;R5B5=1'b Z ;R5B6=1'b
Z ;R5B7=1'b Z ;
//Row 6, BITS 0 TO 7
R6B0=1'b Z;R6B1=1'b Z;R6B2=1'b Z ;R6B3=1'b Z ;R6B4=1'b Z ;R6B5=1'b Z ;R6B6=1'b
Z ;R6B7=1'b Z ;
//Row 7, BITS 0 TO 7
R7B0=1'b Z;R7B1=1'b Z;R7B2=1'b Z ;R7B3=1'b Z ;R7B4=1'b Z ;R7B5=1'b Z ;R7B6=1'b
Z ;R7B7=1'b Z ;
end
endmodule
```

B.4.2 Verilog Code Written to Read an Image From the 8x8 Array

```
//Verilog-AMS HDL for "erosion_DPS", "image_read8" "verilogams"
// Duha Jabakhanji
//June 2 2006
//read in the image value and place in file
'include "constants.vams"
'include "disciplines.vams"
module image_read8 ( R0C0,R0C1,R0C2,R0C3,R0C4,R0C5,R0C6,R0C7,
R1C0,R1C1,R1C2,R1C3,R1C4,R1C5,R1C6,R1C7,
R2C0,R2C1,R2C2,R2C3,R2C4,R2C5,R2C6,R2C7,
R3C0,R3C1,R3C2,R3C3,R3C4,R3C5,R3C6,R3C7,
R4C0,R4C1,R4C2,R4C3,R4C4,R4C5,R4C6,R4C7,
R5C0,R5C1,R5C2,R5C3,R5C4,R5C5,R5C6,R5C7,
R6C0,R6C1,R6C2,R6C3,R6C4,R6C5,R6C6,R6C7,
R7C0,R7C1,R7C2,R7C3,R7C4,R7C5,R7C6,R7C7,
);
input R0C0,R0C1,R0C2,R0C3,R0C4,R0C5,R0C6,R0C7,
R1C0,R1C1,R1C2,R1C3,R1C4,R1C5,R1C6,R1C7,
R2C0,R2C1,R2C2,R2C3,R2C4,R2C5,R2C6,R2C7,
R3C0,R3C1,R3C2,R3C3,R3C4,R3C5,R3C6,R3C7,
R4C0,R4C1,R4C2,R4C3,R4C4,R4C5,R4C6,R4C7,
```

```
R5C0,R5C1,R5C2,R5C3,R5C4,R5C5,R5C6,R5C7,
R6C0,R6C1,R6C2,R6C3,R6C4,R6C5,R6C6,R6C7,
R7C0,R7C1,R7C2,R7C3,R7C4,R7C5,R7C6,R7C7;
reg [7:0]img [0:255];
integer i,j,k;
integer fid;
integer dummy;
initial begin
$shm_open("waves");
$shm_probe("AS");
dummy=0;
#35
//open the file to place the memory in
fid=$fopen ("/home/duha/research/Array8.txt");
for(i=0;i<8;i=i+1)
begin
#0.25
dummy= dummy;
//the cases to mimic the CU
for(j=0;j<8;j=j+1)
begin
#0.25
dummy= dummy;
if(j==0)begin
#2
dummy= dummy;
end else if(j==15)begin
#2 dummy= dummy;
end else begin
#4
dummy= dummy;
end
#4
```

```
k=0;
if(j==0) begin
img[j][i]=R0C0;
k=8;
img[j+k][i]=R1C0;
k=k+8;
img[j+k][i]=R2C0;
k=k+8;
img[j+k][i]=R3C0;
k=k+8;
img[j+k][i]=R4C0;
k=k+8;
img[j+k][i]=R5C0;
k=k+8;
img[j+k][i]=R6C0;
k=k+8;
img[j+k][i]=R7C0;
end
if(j==1) begin
img[j][i]=R0C1;
k=8;
img[j+k][i]=R1C1;
k=k+8;
img[j+k][i]=R2C1;
k=k+8;
img[j+k][i]=R3C1;
k=k+8;
img[j+k][i]=R4C1;
k=k+8;
img[j+k][i]=R5C1;
k=k+8;
img[j+k][i]=R6C1;
k=k+8;
```

```
img[j+k][i]=R7C1;
end
if(j==2) begin
img[j][i]=R0C2;
k=8;
img[j+k][i]=R1C2;
k=k+8;
img[j+k][i]=R2C2;
k=k+8;
img[j+k][i]=R3C2;
k=k+8;
img[j+k][i]=R4C2;
k=k+8;
img[j+k][i]=R5C2;
k=k+8;
img[j+k][i]=R6C2;
k=k+8;
img[j+k][i]=R7C2;
end
if(j==3)begin
img[j][i]=R0C3;
k=8;
img[j+k][i]=R1C3;
k=k+8;
img[j+k][i]=R2C3;
k=k+8;
img[j+k][i]=R3C3;
k=k+8;
img[j+k][i]=R4C3;
k=k+8;
img[j+k][i]=R5C3;
k=k+8;
img[j+k][i]=R6C3;
```

```
k=k+8;
img[j+k][i]=R7C3;
end
if(j==4)begin
img[j][i]=R0C4;
k=8;
img[j+k][i]=R1C4;
k=k+8;
img[j+k][i]=R2C4;
k=k+8;
img[j+k][i]=R3C4;
k=k+8;
img[j+k][i]=R4C4;
k=k+8;
img[j+k][i]=R5C4;
k=k+8;
img[j+k][i]=R6C4;
k=k+8;
img[j+k][i]=R7C4;
end
if(j==5)begin
img[j][i]=R0C5;
k=8;
img[j+k][i]=R1C5;
k=k+8;
img[j+k][i]=R2C5;
k=k+8;
img[j+k][i]=R3C5;
k=k+8;
img[j+k][i]=R4C5;
k=k+8;
img[j+k][i]=R5C5;
k=k+8;
```

```
img[j+k][i]=R6C5;
k=k+8;
img[j+k][i]=R7C5;
end
if(j==6)begin
img[j][i]=R0C6;
k=8;
img[j+k][i]=R1C6;
k=k+8;
img[j+k][i]=R2C6;
k=k+8;
img[j+k][i]=R3C6;
k=k+8;
img[j+k][i]=R4C6;
k=k+8;
img[j+k][i]=R5C6;
k=k+8;
img[j+k][i]=R6C6;
k=k+8;
img[j+k][i]=R7C6;
end
if(j==7)begin
img[j][i]=R0C7;
k=8;
img[j+k][i]=R1C7;
k=k+8;
img[j+k][i]=R2C7;
k=k+8;
img[j+k][i]=R3C7;
k=k+8;
img[j+k][i]=R4C7;
k=k+8;
img[j+k][i]=R5C7;
```

```
k=k+8;
img[j+k][i]=R6C7;
k=k+8;
img[j+k][i]=R7C7;
end
#2
dummy= dummy;
end
end //write to file
for(i=0;i<64;i=i+1) begin
$fwrite(fid,"%h ",img[i]);
end
//close file $fclose(fid);
//used to work, better than the above approach, does not need fopen nor fclose
//$writememh("/home/duha/research/img2.txt",img);
end
endmodule
```

B.4.3 Verilog Code Written to Control the 8x8 Array and Read the Erosion Results

```
//Verilog-AMS HDL for "erosion_DPS", "CU_PE8" "verilogams"
// This is the control unit for the 16X16 array with only the PE
//Functions include:
// Control signals on the array
// Get data from the array and write them to a file
//Finish off the edge detection
'include "constants.vams"
'include "disciplines.vams"
module CU_PE8(
// AEN and NEN as a bus
AEN, NEN,
//Read also as a bus
Read,
```

```
// Tristate control as a bus
NR,NRB,
//The select value in a small bus
S,
//outputs of the array all as buses
//The two domains, D1 and D2
D1, D2 );
//declare the outputs
output [7:0]AEN,NEN,Read,NR, NRB; output [2:0] S;
//declare the inputs
input [7:0] D1,D2;
//internal values
reg [7:0]AEN,NEN,Read,NR,NRB; reg [2:0] S;
//reg [15:0] D1,D2;
//counter values
integer i,j,k;
//file pointers
integer fid1, fid2;
//these hold the output of the array, bit by bit to collect the value
reg [7:0]BitD1 [0:255]; reg [7:0]BitD2 [0:255];
initial begin
$shm_open("waves");
$shm_probe("AS");
//initial state
AEN=0; NEN=0; S=0; Read=0; NR=0; NRB=0;
//Open the files to place the data in
fid1=$fopen ("/home/duha/research/Array8D1.txt");
fid2=$fopen ("/home/duha/research/Array8D2.txt");
//this delay is for the image_gen do not change
#35
//This loop iterates to get the bits
for(i=0;i<8;i=i+1) begin
#0.25
```

```
S=i;
//This loops gets the values one column at a time
for(j=0;j<8;j=j+1) begin
//latch in the domain values
#0.25
//read mem, enable TS, and latch value into A
//read values from the pixels surrounding the current pixel
//this is for the right edge
if(j>0)begin
AEN[j-1]=1;
NRB[j-1]=1;
Read[j-1]=1;
#2 AEN[j-1]=0;
Read[j-1]=0;
end
//this is for the left edge
if(j<7) begin
AEN[j+1]=1;
NRB[j+1]=1;
Read[j+1]=1;
#2
AEN[j+1]=0;
Read[j+1]=0;
end
//now read the current pixel column value
AEN[j]=1;
Read[j]=1;
NR[j]=1;
NEN[j]=1;
NRB[j]=1;
//Read the values in
#4
BitD1[j][i]=D1[0];
```

```
k=8;
BitD1[j+k][i]=D1[1];
k=k+8;
BitD1[j+k][i]=D1[2];
k=k+8;
BitD1[j+k][i]=D1[3];
k=k+8;
BitD1[j+k][i]=D1[4];
k=k+8;
BitD1[j+k][i]=D1[5];
k=k+8;
BitD1[j+k][i]=D1[6];
k=k+8;
BitD1[j+k][i]=D1[7];
//now for D2
BitD2[j][i]=D2[0];
k=8;
BitD2[j+k][i]=D2[1];
k=k+8;
BitD2[j+k][i]=D2[2];
k=k+8;
BitD2[j+k][i]=D2[3];
k=k+8;
BitD2[j+k][i]=D2[4];
k=k+8;
BitD2[j+k][i]=D2[5];
k=k+8;
BitD2[j+k][i]=D2[6];
k=k+8;
BitD2[j+k][i]=D2[7];
#2
//De-assert the signals
AEN[j]=0;
```

```
Read[j]=0;
NR[j]=0;
NEN[j]=0;
NRB[j]=0;
if(j>0)begin
NRB[j-1]=0;
end
if(j<7) begin
NRB[j+1]=0;
end
//end of j (inner) loop
end
//end of i (outer) loop
end
//write the memory to file
for(i=0;i<64;i=i+1) begin
$fwrite(fid1,"%h ",BitD1[i]);
$fwrite(fid2,"%h ",BitD2[i]); end
//close the files
$fclose(fid1);
$fclose(fid2);
//a better way of writing to file, used to work,
//$writememh("/home/duha/research/test2D1.txt",BitD1);
//$writememh("/home/duha/research/test2D2.txt",BitD2);
//end of initial block
end
endmodule
```

B.5 Verilog Code written for the 16x16 Pixel array

B.5.1 Verilog Code Written to Load an Image into the 16x16 Array

```
//Verilog-AMS HDL for "erosion_DPS", "image_gen" "verilogams"
```

```
//This code reads a 16X16 image array from a file and loads it into the pixel
//Duha Jabakhanji
//May 9 2006
//`include "/CMC/tools/cadence/IUS/tools.sun4v/spectre/etc/ahdl/constants.vams"
//`include "/CMC/tools/cadence/IUS/tools.sun4v/spectre/etc/ahdl/disciplines.vams"
module image_gen ( output
//Row 0, BITS 0 TO 7
R0B0,R0B1,R0B2,R0B3,R0B4,R0B5,R0B6,R0B7,
//Row 1, BITS 0 TO 7
R1B0,R1B1,R1B2,R1B3,R1B4,R1B5,R1B6,R1B7,
//Row 2, BITS 0 TO 7
R2B0,R2B1,R2B2,R2B3,R2B4,R2B5,R2B6,R2B7,
//Row 3, BITS 0 TO 7
R3B0,R3B1,R3B2,R3B3,R3B4,R3B5,R3B6,R3B7,
//Row 4, BITS 0 TO 7
R4B0,R4B1,R4B2,R4B3,R4B4,R4B5,R4B6,R4B7,
//Row 5, BITS 0 TO 7
R5B0,R5B1,R5B2,R5B3,R5B4,R5B5,R5B6,R5B7,
//Row 6, BITS 0 TO 7
R6B0,R6B1,R6B2,R6B3,R6B4,R6B5,R6B6,R6B7,
//Row 7, BITS 0 TO 7
R7B0,R7B1,R7B2,R7B3,R7B4,R7B5,R7B6,R7B7,
//Row 8, BITS 0 TO 7
R8B0,R8B1,R8B2,R8B3,R8B4,R8B5,R8B6,R8B7,
//Row 9, BITS 0 TO 7
R9B0,R9B1,R9B2,R9B3,R9B4,R9B5,R9B6,R9B7,
//Row 10, BITS 0 TO 7
R10B0,R10B1,R10B2,R10B3,R10B4,R10B5,R10B6,R10B7,
//Row 11, BITS 0 TO 7
R11B0,R11B1,R11B2,R11B3,R11B4,R11B5,R11B6,R11B7,
//Row 12, BITS 0 TO 7
R12B0,R12B1,R12B2,R12B3,R12B4,R12B5,R12B6,R12B7,
//Row 13, BITS 0 TO 7
```

```
R13B0,R13B1,R13B2,R13B3,R13B4,R13B5,R13B6,R13B7,
//Row 14, BITS 0 TO 7
R14B0,R14B1,R14B2,R14B3,R14B4,R14B5,R14B6,R14B7,
//Row 15, BITS 0 TO 7
R15B0,R15B1,R15B2,R15B3,R15B4,R15B5,R15B6,R15B7,
//WRITE SIGNALS
WC0,WC1,WC2,WC3,WC4,WC5,WC6,WC7,WC8,WC9,WC10,WC11,WC12,WC13,WC14,WC15
);
reg //Row 0, BITS 0 TO 7 R0B0,R0B1,R0B2,R0B3,R0B4,R0B5,R0B6,R0B7,
//Row 1, BITS 0 TO 7
R1B0,R1B1,R1B2,R1B3,R1B4,R1B5,R1B6,R1B7,
//Row 2, BITS 0 TO 7
R2B0,R2B1,R2B2,R2B3,R2B4,R2B5,R2B6,R2B7,
//Row 3, BITS 0 TO 7
R3B0,R3B1,R3B2,R3B3,R3B4,R3B5,R3B6,R3B7,
//Row 4, BITS 0 TO 7
R4B0,R4B1,R4B2,R4B3,R4B4,R4B5,R4B6,R4B7,
//Row 5, BITS 0 TO 7
R5B0,R5B1,R5B2,R5B3,R5B4,R5B5,R5B6,R5B7,
//Row 6, BITS 0 TO 7
R6B0,R6B1,R6B2,R6B3,R6B4,R6B5,R6B6,R6B7,
//Row 7, BITS 0 TO 7
R7B0,R7B1,R7B2,R7B3,R7B4,R7B5,R7B6,R7B7,
//Row 8, BITS 0 TO 7
R8B0,R8B1,R8B2,R8B3,R8B4,R8B5,R8B6,R8B7,
//Row 9, BITS 0 TO 7
R9B0,R9B1,R9B2,R9B3,R9B4,R9B5,R9B6,R9B7,
//Row 10, BITS 0 TO 7
R10B0,R10B1,R10B2,R10B3,R10B4,R10B5,R10B6,R10B7,
//Row 11, BITS 0 TO 7
R11B0,R11B1,R11B2,R11B3,R11B4,R11B5,R11B6,R11B7,
//Row 12, BITS 0 TO 7
R12B0,R12B1,R12B2,R12B3,R12B4,R12B5,R12B6,R12B7,
```

```
//Row 13, BITS 0 TO 7
R13B0,R13B1,R13B2,R13B3,R13B4,R13B5,R13B6,R13B7,
//Row 14, BITS 0 TO 7
R14B0,R14B1,R14B2,R14B3,R14B4,R14B5,R14B6,R14B7,
//Row 15, BITS 0 TO 7
R15B0,R15B1,R15B2,R15B3,R15B4,R15B5,R15B6,R15B7,
WC0,WC1,WC2,WC3,WC4,WC5,WC6,WC7,
WC8,WC9,WC10,WC11,WC12,WC13,WC14,WC15;
integer file, i;
integer p;
reg [7:0]image[0:255];
initial begin
//$shm_open("waves");//$shm_probe("AS");
//initialize the pointers
//position indicates the next spot
//WRITE SIGNALS
WC0=0;WC1=0;WC2=0;WC3=0;WC4=0;WC5=0;WC6=0;WC7=0;
WC8=0;WC9=0;WC10=0;WC11=0;WC12=0;WC13=0;WC14=0;WC15=0;
//open the file to be read
#1 //read the file into the memory
$readmemh("/home/duha/research/test2h.txt",image);
//loop is for the column
for (i=0; i<16; i=i+1) begin
#1
p=16; //Row 0, BITS 0 TO 7
R0B0=image[i][0];R0B1=image[i][1];R0B2=image[i][2];R0B3=image[i][3];
R0B4=image[i][4];R0B5=image[i][5];R0B6=image[i][6];R0B7=image[i][7];
//Row 1, BITS 0 TO 7
R1B0=image[i+p][0];R1B1=image[i+p][1];R1B2=image[i-p][2];R1B3=image[i+p][3];
R1B4=image[i+p][4];R1B5=image[i+p][5];R1B6=image[i-p][6];R1B7=image[i+p][7];
//Row 2, BITS 0 TO 7 p=p+16;
R2B0=image[i+p][0];R2B1=image[i+p][1];R2B2=image[i+p][2];R2B3=image[i+p][3];
R2B4=image[i+p][4];R2B5=image[i+p][5];R2B6=image[i+p][6];R2B7=image[i+p][7];
```

```
//Row 3, BITS 0 TO 7
p=p+16;
R3B0=image[i+p][0];R3B1=image[i+p][1];R3B2=image[i-p][2];R3B3=image[i+p][3];
R3B4=image[i+p][4];R3B5=image[i+p][5];R3B6=image[i-p][6];R3B7=image[i+p][7];
//Row 4, BITS 0 TO 7
p=p+16;
R4B0=image[i+p][0];R4B1=image[i+p][1];R4B2=image[i-p][2];R4B3=image[i+p][3];
R4B4=image[i+p][4];R4B5=image[i+p][5];R4B6=image[i-p][6];R4B7=image[i+p][7];
//Row 5, BITS 0 TO 7
p=p+16;
R5B0=image[i+p][0];R5B1=image[i+p][1];R5B2=image[i-p][2];R5B3=image[i+p][3];
R5B4=image[i+p][4];R5B5=image[i+p][5];R5B6=image[i-p][6];R5B7=image[i+p][7];
//Row 6, BITS 0 TO 7
p=p+16;
R6B0=image[i+p][0];R6B1=image[i+p][1];R6B2=image[i-p][2];R6B3=image[i+p][3];
R6B4=image[i+p][4];R6B5=image[i+p][5];R6B6=image[i-p][6];R6B7=image[i+p][7];
//Row 7, BITS 0 TO 7
p=p+16;
R7B0=image[i+p][0];R7B1=image[i+p][1];R7B2=image[i+p][2];R7B3=image[i+p][3];
R7B4=image[i+p][4];R7B5=image[i+p][5];R7B6=image[i+p][6];R7B7=image[i+p][7];
//Row 8, BITS 0 TO 7
p=p+16;
R8B0=image[i+p][0];R8B1=image[i+p][1];R8B2=image[i-p][2];R8B3=image[i+p][3];
R8B4=image[i+p][4];R8B5=image[i+p][5];R8B6=image[i-p][6];R8B7=image[i+p][7];
//Row 9, BITS 0 TO 7
p=p+16;
R9B0=image[i+p][0];R9B1=image[i+p][1];R9B2=image[i+p][2];R9B3=image[i+p][3];
R9B4=image[i+p][4];R9B5=image[i+p][5];R9B6=image[i+p][6];R9B7=image[i+p][7];
//Row 10, BITS 0 TO 7
p=p+16;
R10B0=image[i+p][0];R10B1=image[i+p][1];R10B2=image[i+p][2];R10B3=image[i+p][3];
R10B4=image[i+p][4];R10B5=image[i+p][5];R10B6=image[i+p][6];R10B7=image[i+p][7];
//Row 11, BITS 0 TO 7
```

```
p=p+16;
R11B0=image[i+p][0];R11B1=image[i+p][1];R11B2=image[i+p][2];R11B3=image[i+p][3];
R11B4=image[i+p][4];R11B5=image[i+p][5];R11B6=image[i+p][6];R11B7=image[i+p][7];
//Row 12, BITS 0 TO 7
p=p+16;
R12B0=image[i+p][0];R12B1=image[i+p][1];R12B2=image[i+p][2];R12B3=image[i+p][3];
R12B4=image[i+p][4];R12B5=image[i+p][5];R12B6=image[i+p][6];R12B7=image[i+p][7];
//Row 13, BITS 0 TO 7
p=p+16;
R13B0=image[i+p][0];R13B1=image[i+p][1];R13B2=image[i+p][2];R13B3=image[i+p][3];
R13B4=image[i+p][4];R13B5=image[i+p][5];R13B6=image[i+p][6];R13B7=image[i+p][7];
//Row 14, BITS 0 TO 7
p=p+16;
R14B0=image[i+p][0];R14B1=image[i+p][1];R14B2=image[i+p][2];R14B3=image[i+p][3];
R14B4=image[i+p][4];R14B5=image[i+p][5];R14B6=image[i+p][6];R14B7=image[i+p][7];
//Row 15, BITS 0 TO 7
p=p+16;
R15B0=image[i+p][0];R15B1=image[i+p][1];R15B2=image[i+p][2];R15B3=image[i+p][3];
R15B4=image[i+p][4];R15B5=image[i+p][5];R15B6=image[i+p][6];R15B7=image[i+p][7];
//assert writes
if(i==0)begin WC0=1; end if(i==1) begin WC1=1; end if(i==2) begin WC2=1; end
if(i==3) begin WC3=1; end if(i==4) begin WC4=1; end if(i==5) begin WC5=1; end
if(i==6) begin WC6=1; end if(i==7) begin WC7=1; end if(i==8) begin WC8=1; end
if(i==9) begin WC9=1; end if(i==10) begin WC10=1; end if(i==11) begin WC11=1; end
if(i==12) begin WC12=1; end if(i==13) begin WC13=1; end if(i==14) begin WC14=1;
end if(i==15) begin WC15=1; end
#2
//deassert reads
WC0=0;WC1=0;WC2=0;WC3=0;WC4=0;WC5=0;WC6=0;WC7=0;
WC8=0;WC9=0;WC10=0;WC11=0;WC12=0;WC13=0;WC14=0;WC15=0;
end
//once done, put the signals in tristate mode
//Row 0, BITS 0 TO 7
```

```
R0B0=1'b Z;R0B1=1'b Z ;R0B2=1'b Z ;R0B3=1'b Z ;R0B4=1'b Z ;R0B5=1'b Z
;R0B6=1'b Z ;R0B7=1'b Z ;
//Row 1, BITS 0 TO 7
R1B0=1'b Z;R1B1=1'b Z;R1B2=1'b Z ;R1B3=1'b Z ;R1B4=1'b Z ;R1B5=1'b Z ;R1B6=1'b
Z ;R1B7=1'b Z ;
//Row 2, BITS 0 TO 7
R2B0=1'b Z;R2B1=1'b Z;R2B2=1'b Z ;R2B3=1'b Z ;R2B4=1'b Z ;R2B5=1'b Z ;R2B6=1'b
Z ;R2B7=1'b Z ;
//Row 3, BITS 0 TO 7
R3B0=1'b Z;R3B1=1'b Z;R3B2=1'b Z ;R3B3=1'b Z ;R3B4=1'b Z ;R3B5=1'b Z ;R3B6=1'b
Z ;R3B7=1'b Z ;
//Row 4, BITS 0 TO 7
R4B0=1'b Z;R4B1=1'b Z;R4B2=1'b Z ;R4B3=1'b Z ;R4B4=1'b Z ;R4B5=1'b Z ;R4B6=1'b
Z ;R4B7=1'b Z ;
//Row 5, BITS 0 TO 7;
R5B0=1'b Z;R5B1=1'b Z;R5B2=1'b Z ;R5B3=1'b Z ;R5B4=1'b Z ;R5B5=1'b Z ;R5B6=1'b
Z ;R5B7=1'b Z ;
//Row 6, BITS 0 TO 7
R6B0=1'b Z;R6B1=1'b Z;R6B2=1'b Z ;R6B3=1'b Z ;R6B4=1'b Z ;R6B5=1'b Z ;R6B6=1'b
Z ;R6B7=1'b Z ;
//Row 7, BITS 0 TO 7
R7B0=1'b Z;R7B1=1'b Z;R7B2=1'b Z ;R7B3=1'b Z ;R7B4=1'b Z ;R7B5=1'b Z ;R7B6=1'b
Z ;R7B7=1'b Z ;
//Row 8, BITS 0 TO 7
R8B0=1'b Z;R8B1=1'b Z;R8B2=1'b Z ;R8B3=1'b Z ;R8B4=1'b Z ;R8B5=1'b Z ;R8B6=1'b
Z ;R8B7=1'b Z ;
//Row 9, BITS 0 TO 7
R9B0=1'b Z;R9B1=1'b Z;R9B2=1'b Z ;R9B3=1'b Z ;R9B4=1'b Z ;R9B5=1'b Z ;R9B6=1'b
Z ;R9B7=1'b Z ;
//Row 10, BITS 0 TO 7
R10B0=1'b Z;R10B1=1'b Z;R10B2=1'b Z ;R10B3=1'b Z ;R10B4=1'b Z ;R10B5=1'b
Z ;R10B6=1'b Z ;R10B7=1'b Z ;
//Row 11, BITS 0 TO 7
```

R11B0=1'b Z;R11B1=1'b Z;R11B2=1'b Z ;R11B3=1'b Z ;R11B4=1'b Z ;R11B5=1'b Z ;R11B6=1'b Z ;R11B7=1'b Z ;

//Row 12, BITS 0 TO 7

R12B0=1'b Z;R12B1=1'b Z;R12B2=1'b Z ;R12B3=1'b Z ;R12B4=1'b Z ;R12B5=1'b Z ;R12B6=1'b Z ;R12B7=1'b Z ;

//Row 13, BITS 0 TO 7

R13B0=1'b Z;R13B1=1'b Z;R13B2=1'b Z ;R13B3=1'b Z ;R13B4=1'b Z ;R13B5=1'b Z ;R13B6=1'b Z ;R13B7=1'b Z ;

//Row 14, BITS 0 TO 7

R14B0=1'b Z;R14B1=1'b Z;R14B2=1'b Z ;R14B3=1'b Z ;R14B4=1'b Z ;R14B5=1'b Z ;R14B6=1'b Z ;R14B7=1'b Z ;

//Row 15, BITS 0 TO 7

R15B0=1'b Z;R15B1=1'b Z;R15B2=1'b Z ;R15B3=1'b Z ;R15B4=1'b Z ;R15B5=1'b Z ;R15B6=1'b Z ;R15B7=1'b Z ;

end

endmodule

B.5.2 Verilog Code Written to Read an Image From the 16x16 Array

```
//Verilog-AMS HDL for "erosion_DPS", "image_read" "verilogams"
// Duha Jabakhanji
//June 2 2006
//read in the image value and place in file
'include "constants.vams" 'include "disciplines.vams"
module image_read (
R0C0,R0C1,R0C2,R0C3,R0C4,R0C5,R0C6,R0C7,
R0C8,R0C9,R0C10,R0C11,R0C12,R0C13,R0C14,R0C15,
R1C0,R1C1,R1C2,R1C3,R1C4,R1C5,R1C6,R1C7,
R1C8,R1C9,R1C10,R1C11,R1C12,R1C13,R1C14,R1C15,
R2C0,R2C1,R2C2,R2C3,R2C4,R2C5,R2C6,R2C7,
R2C8,R2C9,R2C10,R2C11,R2C12,R2C13,R2C14,R2C15,
R3C0,R3C1,R3C2,R3C3,R3C4,R3C5,R3C6,R3C7,
R3C8,R3C9,R3C10,R3C11,R3C12,R3C13,R3C14,R3C15,
```

R4C0,R4C1,R4C2,R4C3,R4C4,R4C5,R4C6,R4C7,
R4C8,R4C9,R4C10,R4C11,R4C12,R4C13,R4C14,R4C15,
R5C0,R5C1,R5C2,R5C3,R5C4,R5C5,R5C6,R5C7,
R5C8,R5C9,R5C10,R5C11,R5C12,R5C13,R5C14,R5C15,
R6C0,R6C1,R6C2,R6C3,R6C4,R6C5,R6C6,R6C7,
R6C8,R6C9,R6C10,R6C11,R6C12,R6C13,R6C14,R6C15,
R7C0,R7C1,R7C2,R7C3,R7C4,R7C5,R7C6,R7C7,
R7C8,R7C9,R7C10,R7C11,R7C12,R7C13,R7C14,R7C15,
R8C0,R8C1,R8C2,R8C3,R8C4,R8C5,R8C6,R8C7,
R8C8,R8C9,R8C10,R8C11,R8C12,R8C13,R8C14,R8C15,
R9C0,R9C1,R9C2,R9C3,R9C4,R9C5,R9C6,R9C7,
R9C8,R9C9,R9C10,R9C11,R9C12,R9C13,R9C14,R9C15,
R10C0,R10C1,R10C2,R10C3,R10C4,R10C5,R10C6,
R10C7,R10C8,R10C9,R10C10,R10C11,R10C12,R10C13,R10C14,R10C15,
R11C0,R11C1,R11C2,R11C3,R11C4,R11C5,R11C6,
R11C7,R11C8,R11C9,R11C10,R11C11,R11C12,R11C13,R11C14,R11C15,
R12C0,R12C1,R12C2,R12C3,R12C4,R12C5,R12C6,
R12C7,R12C8,R12C9,R12C10,R12C11,R12C12,R12C13,R12C14,R12C15,
R13C0,R13C1,R13C2,R13C3,R13C4,R13C5,R13C6,
R13C7,R13C8,R13C9,R13C10,R13C11,R13C12,R13C13,R13C14,R13C15,
R14C0,R14C1,R14C2,R14C3,R14C4,R14C5,R14C6,
R14C7,R14C8,R14C9,R14C10,R14C11,R14C12,R14C13,R14C14,R14C15,
R15C0,R15C1,R15C2,R15C3,R15C4,R15C5,R15C6,
R15C7,R15C8,R15C9,R15C10,R15C11,R15C12,R15C13,R15C14,R15C15
);
input
R0C0,R0C1,R0C2,R0C3,R0C4,R0C5,R0C6,R0C7,
R0C8,R0C9,R0C10,R0C11,R0C12,R0C13,R0C14,R0C15,
R1C0,R1C1,R1C2,R1C3,R1C4,R1C5,R1C6,R1C7,
R1C8,R1C9,R1C10,R1C11,R1C12,R1C13,R1C14,R1C15,
R2C0,R2C1,R2C2,R2C3,R2C4,R2C5,R2C6,R2C7,
R2C8,R2C9,R2C10,R2C11,R2C12,R2C13,R2C14,R2C15,
R3C0,R3C1,R3C2,R3C3,R3C4,R3C5,R3C6,R3C7,

```
R3C8,R3C9,R3C10,R3C11,R3C12,R3C13,R3C14,R3C15,
R4C0,R4C1,R4C2,R4C3,R4C4,R4C5,R4C6,R4C7,
R4C8,R4C9,R4C10,R4C11,R4C12,R4C13,R4C14,R4C15,
R5C0,R5C1,R5C2,R5C3,R5C4,R5C5,R5C6,R5C7,
R5C8,R5C9,R5C10,R5C11,R5C12,R5C13,R5C14,R5C15,
R6C0,R6C1,R6C2,R6C3,R6C4,R6C5,R6C6,R6C7,
R6C8,R6C9,R6C10,R6C11,R6C12,R6C13,R6C14,R6C15,
R7C0,R7C1,R7C2,R7C3,R7C4,R7C5,R7C6,R7C7,
R7C8,R7C9,R7C10,R7C11,R7C12,R7C13,R7C14,R7C15,
R8C0,R8C1,R8C2,R8C3,R8C4,R8C5,R8C6,R8C7,
R8C8,R8C9,R8C10,R8C11,R8C12,R8C13,R8C14,R8C15,
R9C0,R9C1,R9C2,R9C3,R9C4,R9C5,R9C6,R9C7,
R9C8,R9C9,R9C10,R9C11,R9C12,R9C13,R9C14,R9C15,
R10C0,R10C1,R10C2,R10C3,R10C4,R10C5,R10C6,R10C7,
R10C8,R10C9,R10C10,R10C11,R10C12,R10C13,R10C14,R10C15,
R11C0,R11C1,R11C2,R11C3,R11C4,R11C5,R11C6,R11C7,
R11C8,R11C9,R11C10,R11C11,R11C12,R11C13,R11C14,R11C15,
R12C0,R12C1,R12C2,R12C3,R12C4,R12C5,R12C6,R12C7,
R12C8,R12C9,R12C10,R12C11,R12C12,R12C13,R12C14,R12C15,
R13C0,R13C1,R13C2,R13C3,R13C4,R13C5,R13C6,R13C7,
R13C8,R13C9,R13C10,R13C11,R13C12,R13C13,R13C14,R13C15,
R14C0,R14C1,R14C2,R14C3,R14C4,R14C5,R14C6,R14C7,
R14C8,R14C9,R14C10,R14C11,R14C12,R14C13,R14C14,R14C15,
R15C0,R15C1,R15C2,R15C3,R15C4,R15C5,R15C6,R15C7,
R15C8,R15C9,R15C10,R15C11,R15C12,R15C13,R15C14,R15C15;
reg [7:0]img [0:255];
reg dummy; integer i,j,k;
integer fid;
initial begin
$shm_open("waves");
$shm_probe("AS");
dummy=0;
#55
```

```
//open the file to place the memory in
fid=$fopen ("/home/duha/research/img2.txt");
for(i=0;i<8;i=i+1) begin
#0.25
dummy= dummy;
//the cases to mimic the CU
for(j=0;j<16;j=j+1) begin
#0.25
dummy= dummy;
if(j==0)begin
#2
dummy= dummy;
end else
if(j==15)begin
#2
dummy= dummy;
end else
begin
#4
dummy= dummy;
end
#6
k=0;
if(j==0) begin img[j][i]=R0C0; k=16; img[j+k][i]=R1C0; k=k+16; img[j+k][i]=R2C0;
k=k+16; img[j+k][i]=R3C0; k=k+16; img[j+k][i]=R4C0; k=k+16; img[j+k][i]=R5C0;
k=k+16; img[j+k][i]=R6C0; k=k+16; img[j+k][i]=R7C0; k=k+16; img[j+k][i]=R8C0;
k=k+16; img[j+k][i]=R9C0; k=k+16; img[j+k][i]=R10C0; k=k+16; img[j+k][i]=R11C0;
k=k+16; img[j+k][i]=R12C0; k=k+16; img[j+k][i]=R13C0; k=k+16; img[j+k][i]=R14C0;
k=k+16; img[j+k][i]=R15C0;
end
if(j==1) begin img[j][i]=R0C1; k=16; img[j+k][i]=R1C1; k=k+16; img[j+k][i]=R2C1;
k=k+16; img[j+k][i]=R3C1; k=k+16; img[j+k][i]=R4C1; k=k+16; img[j+k][i]=R5C1;
k=k+16; img[j+k][i]=R6C1; k=k+16; img[j+k][i]=R7C1; k=k+16; img[j+k][i]=R8C1;
```

k=k+16; img[j+k][i]=R9C1; k=k+16; img[j+k][i]=R10C1; k=k+16; img[j+k][i]=R11C1; k=k+16; img[j+k][i]=R12C1; k=k+16; img[j+k][i]=R13C1; k=k+16; img[j+k][i]=R14C1; k=k+16; img[j+k][i]=R15C1;

end

if(j==2) begin img[j][i]=R0C2; k=16; img[j+k][i]=R1C2; k=k+16; img[j+k][i]=R2C2; k=k+16; img[j+k][i]=R3C2; k=k+16; img[j+k][i]=R4C2; k=k+16; img[j+k][i]=R5C2; k=k+16; img[j+k][i]=R6C2; k=k+16; img[j+k][i]=R7C2; k=k+16; img[j+k][i]=R8C2; k=k+16; img[j+k][i]=R9C2; k=k+16; img[j+k][i]=R10C2; k=k+16; img[j+k][i]=R11C2; k=k+16; img[j+k][i]=R12C2; k=k+16; img[j+k][i]=R13C2; k=k+16; img[j+k][i]=R14C2; k=k+16; img[j+k][i]=R15C2;

end

if(j==3)begin img[j][i]=R0C3; k=16; img[j+k][i]=R1C3; k=k+16; img[j+k][i]=R2C3; k=k+16; img[j+k][i]=R3C3; k=k+16; img[j+k][i]=R4C3; k=k+16; img[j+k][i]=R5C3; k=k+16; img[j+k][i]=R6C3; k=k+16; img[j+k][i]=R7C3; k=k+16; img[j+k][i]=R8C3; k=k+16; img[j+k][i]=R9C3; k=k+16; img[j+k][i]=R10C3; k=k+16; img[j+k][i]=R11C3; k=k+16; img[j+k][i]=R12C3; k=k+16; img[j+k][i]=R13C3; k=k+16; img[j+k][i]=R14C3; k=k+16; img[j+k][i]=R15C3;

end

if(j==4)begin img[j][i]=R0C4; k=16; img[j+k][i]=R1C4; k=k+16; img[j+k][i]=R2C4; k=k+16; img[j+k][i]=R3C4; k=k+16; img[j+k][i]=R4C4; k=k+16; img[j+k][i]=R5C4; k=k+16; img[j+k][i]=R6C4; k=k+16; img[j+k][i]=R7C4; k=k+16; img[j+k][i]=R8C4; k=k+16; img[j+k][i]=R9C4; k=k+16; img[j+k][i]=R10C4; k=k+16; img[j+k][i]=R11C4; k=k+16; img[j+k][i]=R12C4; k=k+16; img[j+k][i]=R13C4; k=k+16; img[j+k][i]=R14C4; k=k+16; img[j+k][i]=R15C4;

end

if(j==5)begin img[j][i]=R0C5; k=16; img[j+k][i]=R1C5; k=k+16; img[j+k][i]=R2C5; k=k+16; img[j+k][i]=R3C5; k=k+16; img[j+k][i]=R4C5; k=k+16; img[j+k][i]=R5C5; k=k+16; img[j+k][i]=R6C5; k=k+16; img[j+k][i]=R7C5; k=k+16; img[j+k][i]=R8C5; k=k+16; img[j+k][i]=R9C5; k=k+16; img[j+k][i]=R10C5; k=k+16; img[j+k][i]=R11C5; k=k+16; img[j+k][i]=R12C5; k=k+16; img[j+k][i]=R13C5; k=k+16; img[j+k][i]=R14C5; k=k+16; img[j+k][i]=R15C5;

end

if(j==6)begin img[j][i]=R0C6; k=16; img[j+k][i]=R1C6; k=k+16; img[j+k][i]=R2C6;

```
k=k+16; img[j+k][i]=R3C6; k=k+16; img[j+k][i]=R4C6; k=k+16; img[j+k][i]=R5C6;
k=k+16; img[j+k][i]=R6C6; k=k+16; img[j+k][i]=R7C6; k=k+16; img[j+k][i]=R8C6;
k=k+16; img[j+k][i]=R9C6; k=k+16; img[j+k][i]=R10C6; k=k+16; img[j+k][i]=R11C6;
k=k+16; img[j+k][i]=R12C6; k=k+16; img[j+k][i]=R13C6; k=k+16; img[j+k][i]=R14C6;
k=k+16; img[j+k][i]=R15C6;
    end
    if(j==7)begin img[j][i]=R0C7; k=16; img[j+k][i]=R1C7; k=k+16; img[j+k][i]=R2C7;
k=k+16; img[j+k][i]=R3C7; k=k+16; img[j+k][i]=R4C7; k=k+16; img[j+k][i]=R5C7;
k=k+16; img[j+k][i]=R6C7; k=k+16; img[j+k][i]=R7C7; k=k+16; img[j+k][i]=R8C7;
k=k+16; img[j+k][i]=R9C7; k=k+16; img[j+k][i]=R10C7; k=k+16; img[j+k][i]=R11C7;
k=k+16; img[j+k][i]=R12C7; k=k+16; img[j+k][i]=R13C7; k=k+16; img[j+k][i]=R14C7;
k=k+16; img[j+k][i]=R15C7;
    end
    if(j==8)begin img[j][i]=R0C8; k=16; img[j+k][i]=R1C8; k=k+16; img[j+k][i]=R2C8;
k=k+16; img[j+k][i]=R3C8; k=k+16; img[j+k][i]=R4C8; k=k+16; img[j+k][i]=R5C8;
k=k+16; img[j+k][i]=R6C8; k=k+16; img[j+k][i]=R7C8; k=k+16; img[j+k][i]=R8C8;
k=k+16; img[j+k][i]=R9C8; k=k+16; img[j+k][i]=R10C8; k=k+16; img[j+k][i]=R11C8;
k=k+16; img[j+k][i]=R12C8; k=k+16; img[j+k][i]=R13C8; k=k+16; img[j+k][i]=R14C8;
k=k+16; img[j+k][i]=R15C8;
    end
    if(j==9)begin img[j][i]=R0C9; k=16; img[j+k][i]=R1C9; k=k+16; img[j+k][i]=R2C9;
k=k+16; img[j+k][i]=R3C9; k=k+16; img[j+k][i]=R4C9; k=k+16; img[j+k][i]=R5C9;
k=k+16; img[j+k][i]=R6C9; k=k+16; img[j+k][i]=R7C9; k=k+16; img[j+k][i]=R8C9;
k=k+16; img[j+k][i]=R9C9; k=k+16; img[j+k][i]=R10C9; k=k+16; img[j+k][i]=R11C9;
k=k+16; img[j+k][i]=R12C9; k=k+16; img[j+k][i]=R13C9; k=k+16; img[j+k][i]=R14C9;
k=k+16; img[j+k][i]=R15C9;
    end
    if(j==10)begin img[j][i]=R0C10; k=16; img[j+k][i]=R1C10; k=k+16; img[j+k][i]=R2C10;
k=k+16; img[j+k][i]=R3C10; k=k+16; img[j+k][i]=R4C10; k=k+16; img[j+k][i]=R5C10;
k=k+16; img[j+k][i]=R6C10; k=k+16; img[j+k][i]=R7C10; k=k+16; img[j+k][i]=R8C10;
k=k+16; img[j+k][i]=R9C10; k=k+16; img[j+k][i]=R10C10; k=k+16; img[j+k][i]=R11C10;
k=k+16; img[j+k][i]=R12C10; k=k+16; img[j+k][i]=R13C10; k=k+16; img[j+k][i]=R14C10;
k=k+16; img[j+k][i]=R15C10;
```

```
    end
    if(j==11)begin img[j][i]=R0C11; k=16; img[j+k][i]=R1C11; k=k+16; img[j+k][i]=R2C11;
k=k+16; img[j+k][i]=R3C11; k=k+16; img[j+k][i]=R4C11; k=k+16; img[j+k][i]=R5C11;
k=k+16; img[j+k][i]=R6C11; k=k+16; img[j+k][i]=R7C11; k=k+16; img[j+k][i]=R8C11;
k=k+16; img[j+k][i]=R9C11; k=k+16; img[j+k][i]=R10C11; k=k+16; img[j+k][i]=R11C11;
k=k+16; img[j+k][i]=R12C11; k=k+16; img[j+k][i]=R13C11; k=k+16; img[j+k][i]=R14C11;
k=k+16; img[j+k][i]=R15C11;
    end
    if(j==12)begin img[j][i]=R0C12; k=16; img[j+k][i]=R1C12; k=k+16; img[j+k][i]=R2C12;
k=k+16; img[j+k][i]=R3C12; k=k+16; img[j+k][i]=R4C12; k=k+16; img[j+k][i]=R5C12;
k=k+16; img[j+k][i]=R6C12; k=k+16; img[j+k][i]=R7C12; k=k+16; img[j+k][i]=R8C12;
k=k+16; img[j+k][i]=R9C12; k=k+16; img[j+k][i]=R10C12; k=k+16; img[j+k][i]=R11C12;
k=k+16; img[j+k][i]=R12C12; k=k+16; img[j+k][i]=R13C12; k=k+16; img[j+k][i]=R14C12;
k=k+16; img[j+k][i]=R15C12;
    end
    if(j==13)begin img[j][i]=R0C13; k=16; img[j+k][i]=R1C13; k=k+16; img[j+k][i]=R2C13;
k=k+16; img[j+k][i]=R3C13; k=k+16; img[j+k][i]=R4C13; k=k+16; img[j+k][i]=R5C13;
k=k+16; img[j+k][i]=R6C13; k=k+16; img[j+k][i]=R7C13; k=k+16; img[j+k][i]=R8C13;
k=k+16; img[j+k][i]=R9C13; k=k+16; img[j+k][i]=R10C13; k=k+16; img[j+k][i]=R11C13;
k=k+16; img[j+k][i]=R12C13; k=k+16; img[j+k][i]=R13C13; k=k+16; img[j+k][i]=R14C13;
k=k+16; img[j+k][i]=R15C13;
    end
    if(j==14)begin img[j][i]=R0C14; k=16; img[j+k][i]=R1C14; k=k+16; img[j+k][i]=R2C14;
k=k+16; img[j+k][i]=R3C14; k=k+16; img[j+k][i]=R4C14; k=k+16; img[j+k][i]=R5C14;
k=k+16; img[j+k][i]=R6C14; k=k+16; img[j+k][i]=R7C14; k=k+16; img[j+k][i]=R8C14;
k=k+16; img[j+k][i]=R9C14; k=k+16; img[j+k][i]=R10C14; k=k+16; img[j+k][i]=R11C14;
k=k+16; img[j+k][i]=R12C14; k=k+16; img[j+k][i]=R13C14; k=k+16; img[j+k][i]=R14C14;
k=k+16; img[j+k][i]=R15C14;
    end
    if(j==15)begin img[j][i]=R0C15; k=16; img[j+k][i]=R1C15; k=k+16; img[j+k][i]=R2C15;
k=k+16; img[j+k][i]=R3C15; k=k+16; img[j+k][i]=R4C15; k=k+16; img[j+k][i]=R5C15;
k=k+16; img[j+k][i]=R6C15; k=k+16; img[j+k][i]=R7C15; k=k+16; img[j+k][i]=R8C15;
k=k+16; img[j+k][i]=R9C15; k=k+16; img[j+k][i]=R10C15; k=k+16; img[j+k][i]=R11C15;
```

```
k=k+16; img[j+k][i]=R12C15; k=k+16; img[j+k][i]=R13C15; k=k+16; img[j+k][i]=R14C15;
k=k+16; img[j+k][i]=R15C15;
    end
    #2
    dummy= dummy;
    end
    end
    //write to file
    for(i=0;i<256;i=i+1) begin
    $fwrite(fid,"%h ",img[i]);
    end
    //close file
    $fclose(fid);
    //used to work, better than the above approach, does not need fopen nor fclose
    //$writememh("/home/duha/research/img2.txt",img);
    end
    endmodule
```

B.5.3 Verilog Code Written to Control the 16x16 Array and Read the Erosion Results

```
//Verilog-AMS HDL for "erosion_DPS", "CU_PE16" "verilogams"
    // This is the control unit for the 16X16 array with only the PE
    //Functions include:
    // Control signals on the array
    // Get data from the array and write them to a file
    //Finish off the edge detection
    'include "constants.vams" 'include "disciplines.vams"
    module CU_PE16 (
    // AEN and NEN as a bus
    AEN, NEN,
    //Read also as a bus
    Read,
    // Tristate control as a bus
```

```
NR,NRB,
//The select value in a small bus
S,
//outputs of the array all as buses
//The two domains, D1 and D2
D1, D2 );
//declare the outputs
output [15:0]AEN,NEN,Read,NR, NRB; output [2:0] S;
//declare the inputs
input [15:0] D1,D2;
//internal values
reg [15:0]AEN,NEN,Read,NR,NRB; reg [2:0] S;
//reg [15:0] D1,D2;
//counter values
integer i,j,k,d;
//file pointers
integer fid1, fid2;
//these hold the output of the array, bit by bit to collect the value
reg [7:0]BitD1 [0:255]; reg [7:0]BitD2 [0:255];
initial begin
$shm_open("waves"); $shm_probe("AS"); //initial state
AEN=0; NEN=0; S=0; Read=0; NR=0; NRB=0;
//Open the files to place the data in
fid1=$fopen ("/home/duha/research/test2D1.txt");
fid2=$fopen ("/home/duha/research/test2D2.txt");
//this delay is for the image_gen do not change
#55
//This loop iterates to get the bits
for(i=0;i<8;i=i+1) begin
#0.25
S=i;
//This loops gets the values one column at a time
for(j=0;j<16;j=j+1) begin
```

```
//latch in the domain values
#0.25
//read mem, enable TS, and latch value into A
//read values from the pixels surrounding the current pixel
//this is for the right edge
if(j>0)begin
AEN[j-1]=1;
NRB[j-1]=1;
Read[j-1]=1;
#2
AEN[j-1]=0;
Read[j-1]=0;
end //this is for the left edge
if(j<15) begin
AEN[j+1]=1;
NRB[j+1]=1;
Read[j+1]=1;
#2
AEN[j+1]=0;
Read[j+1]=0;
end
//now read the current pixel column value
AEN[j]=1;
Read[j]=1;
NR[j]=1;
NEN[j]=1;
NRB[j]=1;
//Read the values in
#6
BitD1[j][i]=D1[0]; k=16; BitD1[j+k][i]=D1[1]; k=k+16; BitD1[j+k][i]=D1[2]; k=k+16;
BitD1[j+k][i]=D1[3]; k=k+16; BitD1[j+k][i]=D1[4]; k=k+16; BitD1[j+k][i]=D1[5]; k=k+16;
BitD1[j+k][i]=D1[6]; k=k+16; BitD1[j+k][i]=D1[7]; k=k+16; BitD1[j+k][i]=D1[8]; k=k+16;
BitD1[j+k][i]=D1[9]; k=k+16; BitD1[j+k][i]=D1[10]; k=k+16; BitD1[j+k][i]=D1[11]; k=k+16;
```

```
BitD1[j+k][i]=D1[12]; k=k+16; BitD1[j+k][i]=D1[13]; k=k+16; BitD1[j+k][i]=D1[14]; k=k+16;
BitD1[j+k][i]=D1[15];
    //now for D2
    BitD2[j][i]=D2[0]; k=16; BitD2[j+k][i]=D2[1]; k=k+16; BitD2[j+k][i]=D2[2]; k=k+16;
BitD2[j+k][i]=D2[3]; k=k+16; BitD2[j+k][i]=D2[4]; k=k+16; BitD2[j+k][i]=D2[5]; k=k+16;
BitD2[j+k][i]=D2[6]; k=k+16; BitD2[j+k][i]=D2[7]; k=k+16; BitD2[j+k][i]=D2[8]; k=k+16;
BitD2[j+k][i]=D2[9]; k=k+16; BitD2[j+k][i]=D2[10]; k=k+16; BitD2[j+k][i]=D2[11]; k=k+16;
BitD2[j+k][i]=D2[12]; k=k+16; BitD2[j+k][i]=D2[13]; k=k+16; BitD2[j+k][i]=D2[14]; k=k+16;
BitD2[j+k][i]=D2[15];
    #2
    //De-assert the signals
    AEN[j]=0;
    Read[j]=0;
    NR[j]=0;
    NEN[j]=0;
    NRB[j]=0;
    if(j>0)begin
    NRB[j-1]=0;
    end
    if(j<15) begin
    NRB[j+1]=0;
    end
    //end of j (inner) loop
    end
    //end of i (outer) loop
    end
    //write the memory to file
    for(i=0;i¡256;i=i+1) begin
    $fwrite(fid1,"%h ",BitD1[i]);
    $fwrite(fid2,"%h ",BitD2[i]);
    end
    //close the files
    $fclose(fid1); $fclose(fid2);
```

```
//a better way of writing to file, used to work,
//$writememh("/home/duha/research/test2D1.txt",BitD1);
//$writememh("/home/duha/research/test2D2.txt",BitD2);
//end of initial block
end
endmodule
```

B.6 Verilog Code written for a Pixel Array Control Unit Ready for synthesis

```
//Verilog-AMS HDL for "Arrays", "CU" "verilogams"
    //Duha Jabakhanji
    //this is a synthasizable cu for a 16x16 array with pd
    //`include "constants.vams" //`include "disciplines.vams"
    module CU (
    //Pixel signals
    VgpO,VgnO,
    //Tristate buffers
    W_TSO,NW_TSO,
    //Bias ckt
    PEO,NPEO,
    //MUXS
    SO,
    //OUTPUTS
    D1, D2,
    //cnt
    enO, rstO,
    // AEN and NEN as a bus
    AENO, NENO,
    //Read also as a bus
    ReadO,
    // Tristate control as a bus
    NRO,NRBO,
```

```
//clock
clk,
//reset for cu
reset
);
//*** for synthesis comment out***
initial begin $shm_open("mywaves"); $shm_probe("AS"); end
//      Outputs
//declare the outputs
output [15:0]AENO,NENO,ReadO,NRO, NRBO; output [2:0] SO; output
//Pixel signals
VgpO,VgnO,
//Tristate buffers
W_TSO,NW_TSO,
//Bias ckt
PEO,NPEO;
output enO, rstO;
//       inputs
input [15:0] D1,D2;
input clk,reset;
//      internal values
reg [15:0]AEN,NEN,Read,NR,NRB;
reg [2:0] S;
reg[2:0] state, next_state,last_state;
reg [7:0]BitD1 [0:255];
reg [7:0]BitD2 [0:255];
reg //Tristate buffers
W_TS,NW_TS,
//Bias ckt
PE,NPE,
Vgp,Vgn,en, rst;
reg [3:0]i,j;
reg [7:0]k;
```

```
reg [15:0] cnt,ptr;
wire [15:0]AEN0,NEN0,Read0,NR0,NRB0;
wire[2:0] SO;
wire //Tristate buffers
W_TS0,NW_TS0,
//Bias ckt
PE0,NPE0,
Vgp0,Vgn0,en0, rst0;
//      -Parameters
parameter [2:0] S0=3'b000; //reset state
parameter [2:0] S1=3'b001; //integration state
parameter [2:0] S2=3'b010; //ADC state
parameter [2:0] S3=3'b011; //erosion state
parameter [2:0] S4=3'b100;
parameter [2:0] S5=3'b101;
parameter [2:0] S6=3'b110;
parameter [2:0] S7=3'b111;
//          code starts here
always @(posedge clk) begin
if (reset) begin
next_state=S0;
cnt=0;
i=0;
j=0;
end
cnt=cnt+1;
end
always @(posedge clk) begin
if (reset)
begin next_state=S0;
last_state=S0;
cnt=0;
i=0;
```

```
j=0;
end
state=next_state;
//case statment to go through the states
case(state)
//reset state
S0:begin
S=0;
NPE=1; PE=0;
NW_TS=1; W_TS=0;
Vgp=1; Vgn=0;
rst = 1; en = 1;
AEN=0; NEN=0; S=0; Read=0; NR=0; NRB=0;
i=0;
j=0;
if(last_state==S7)begin
cnt=0;
last_state=S0;
end
if(cnt>=3)begin
next_state=S1;
end
else begin
next_state=S0;
end
end
//integration state
S1:begin
rst = 0;
Vgp=0; Vgn=0;
NW_TS=1; W_TS=0;
NPE=0; PE=1;
if (cnt>=97)begin
```

```
next_state=S2;
end else begin
next_state=S1;
end
end
//ADC
S2:begin
Vgp=0; Vgn=0;
NPE=0; PE=1;
NW_TS=0; W_TS=1;
if(cnt>=20000)begin
next_state=S3;
end else begin
next_state=S2;
end
end
//Erosion
S3:begin
S=i;
//This loops gets the values one column at a time
//latch in the domain values
//read mem, enable TS, and latch value into A
//read values from the pixels surrounding the current pixel
//this is for the right edge
if(j>0)begin
AEN[j-1]=1;
NRB[j-1]=1;
Read[j-1]=1;
end
next_state=S4;
end
S4:begin
if(j>0)begin
```

```
AEN[j-1]=0;
Read[j-1]=0;
end
//this is for the left edge
if(j<15) begin
AEN[j+1]=1;
NRB[j+1]=1;
Read[j+1]=1;
end next_state=S5;
end
S5:begin
if(j<15) begin
AEN[j+1]=0;
Read[j+1]=0;
end
//now read the current pixel column value
AEN[j]=1;
Read[j]=1;
NR[j]=1;
NEN[j]=1;
NRB[j]=1;
ptr=cnt+3;
next_state=S6;
end
S6: begin
if(cnt>=ptr)begin
BitD1[j][i]=D1[0]; //k=16; BitD1[j+16][i]=D1[1];
//k=k+16; BitD1[j+32][i]=D1[2]; //k=k+16; BitD1[j+48][i]=D1[3];
//k=k+16; BitD1[j+64][i]=D1[4]; //k=k+16; BitD1[j+80][i]=D1[5];
//k=k+16; BitD1[j+96][i]=D1[6]; //k=k+16; BitD1[j+112][i]=D1[7];
//k=k+16; BitD1[j+128][i]=D1[8]; //k=k+16; BitD1[j+144][i]=D1[9];
//k=k+16; BitD1[j+160][i]=D1[10]; //k=k+16; BitD1[j+176][i]=D1[11];
//k=k+16; BitD1[j+192][i]=D1[12]; //k=k+16; BitD1[j+208][i]=D1[13];
```

```
//k=k+16; BitD1[j+224][i]=D1[14]; //k=k+16; BitD1[j+240][i]=D1[15];
//now for D2
BitD2[j][i]=D2[0]; //k=16; BitD2[j+16][i]=D2[1];
//k=k+16; BitD2[j+32][i]=D2[2]; //k=k+16; BitD2[j+48][i]=D2[3];
//k=k+16; BitD2[j+64][i]=D2[4]; //k=k+16; BitD2[j+80][i]=D2[5];
//k=k+16; BitD2[j+96][i]=D2[6]; //k=k+16; BitD2[j+112][i]=D2[7];
//k=k+16; BitD2[j+128][i]=D2[8]; //k=k+16; BitD2[j+144][i]=D2[9];
//k=k+16; BitD2[j+160][i]=D2[10]; //k=k+16; BitD2[j+176][i]=D2[11];
//k=k+16; BitD2[j+192][i]=D2[12]; //k=k+16; BitD2[j+208][i]=D2[13];
//k=k+16; BitD2[j+224][i]=D2[14]; //k=k+16; BitD2[j+240][i]=D2[15];
next_state=S7;
end else begin
next_state=S6;
end
end
S7:begin //De-assert the signals
AEN[j]=0;
Read[j]=0;
NR[j]=0;
NEN[j]=0;
NRB[j]=0;
if(j>0)begin
NRB[j-1]=0;
end
if(j<15) begin
NRB[j+1]=0;
end
if(j==15)begin
i=i+1; j=0;
end if(i>=8)
begin next_state=S0;
last_state=S7;
end else
```

```
begin next_state=S3;
end
j=j+1;
//end of last case
end
//end of case statment
endcase
//end of always
end
assign AEN0=AEN;
assign NEN0=NEN;
assign Read0=Read;
assign NR0=NR;
assign NRB0=NRB;
assign SO=S;
assign W_TS0=W_TS;
assign NW_TS0=NW_TS;
assign PE0=PE;
assign NPE0=NPE;
assign Vgp0=Vgp;
assign Vgn0=Vgn;
assign en0=en;
assign rst0=rst;
endmodule
```

Bibliography

[1] A. Trepanier, J.-L. Trepanier, M. Sawan, and Y. Audet, "A multiple operations mode cmos digital pixel sensor dedicated to a visual cortical implant," *The 47th IEEE International Midwest Symposium on circuits and Systems*, pp. I–369–372, 2004.

[2] J.-L. Trepanier, M. Sawan, and Y. Audet, "A new CMOS architecture for wide dynamic range image sensing," *Canadian Conference on Electrical and Computer Engineering 2003 - CCGEI2003*, pp. 323–326, May 2003.

[3] S. Kleinfelder, S. Lim, X. Liu, and A. E. Gamal, "A 10, 000 frames/s CMOS digital pixel sensor," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 12, pp. 2049–2059, December 2001.

[4] A. Kitchen, A. Bermak, and A. Bouzerdoum, "PWM digital pixel sensor based on asynchronous self-reseting scheme," *IEEE Electronic Device Letters*, vol. 25, no. 7, pp. 471–473, July 2004.

[5] A. Bermark and A. Kitchen, "A novel adaptive logarithmic digital pixel sensor," *IEEE Photonics Technology Letters*, vol. 18, no. 20, pp. 2147–2149, October 2006.

[6] A. Bermark and Y.-F. Yung, "A DPS array with programmable resolution and reconfigurable conversion time," *IEEE Transaction on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 1, pp. 15–22, January 2006.

[7] R. E.-C. E. Culurciello and K. Boahen, "A biomorphic digital image sensor," *IEEE Journal of Solid-State Electronics*, vol. 38, no. 2, February 2003.

[8] M. Mazza, P. Renaud, D. C. Bertrand, and A. M. Ionescu, "CMOS pixels for subretinal implantable prosthesis," *IEEE Sensors Journal*, vol. 5, no. 1, pp. 32–37, February 2005.

[9] S. Kleinfelder, S. Lim, X. Liu, and A. E. Gamal, "CMOS image sensor with embedded processors," *IEEE International Solid State Circuits Conference*, pp. 433–436, 2001.

[10] A. O. Ercan, F. Xiao, B. Wandell, S. Lim, X. Liu, and A. E. Gamal, "Experimental high speed CMOS image sensor system and applications," *Proceedings of the IEEE Sensors*, vol. 1, pp. 12–20, June 2002.

[11] S. Kavusi and A. ElGamal, "Quantitative study of high dynamic range image sensor architectures," *Proceedings of SPIE-IST Electronic Imaging*, vol. 5301, pp. 264–275, 2004.

[12] S. Kagami, T. Komuro, and M. Ishikawa, "An advanced digital vision chip and its system implementation," *SICE Annual Conference in Fukui*, pp. 1568–1571, August 2003.

[13] D. Fey, L. Hoppe and A. Loos, "Reconfigurable on-chip SIMD processor architectures for intelligent CMOS camera chips," *IEEE International Conference on Parallel Computing in Electrical Engineering*, 2004.

[14] T. Hammadou, M. Nilson, A. Bermak, and P. Ogunbona, "A 96x64 intelligent digital pixel array with extended binary stochastic arithmetic," *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, pp. 772–775, May 2003.

[15] J. Zhang and C. Wang, "Design of a sensor array circuit for edge detection," *Proceedings of IEEE Sensors*, vol. 1, pp. 219–222, October 2004.

[16] M. Barbaro, P.-Y. Burgi, A. Mortara, P. Nussbaum, and F. Heitger, "A 100 by 100 pixel silicon retina for gradient extraction with steering filter capabilities and temporal output coding," *IEEE Journal of Solid-State Electronics*, vol. 37, no. 2, February 2002.

[17] S. Kagami, T. Komuro, and M. Ishikawa, "A reat-time visual processing system using a general-purpose vision chip," *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, pp. 1229–1234, May 2002.

[18] S. Kagami, T. Komuro, and M. Ishikawa, "A high-speed vision system with in-pixel programmable ADCs and PEs for real-time visual sensing," *AMC 2004 Kawasaki Japan*, pp. 439–443, 2004.

[19] S. Kagami, T. Komuro, and M. Ishikawa, "Digital vision chips and high-speed vision systems," *Symposium on VLSI Circuits Digest of Technical Papers*, pp. 1–4, 2001.

[20] S. Kagami, T. Komuro, and M. Ishikawa, "A dynamically reconfigurable SIMD processor for a vision chip," *IEEE Journal of Solid State Circuits*, vol. 39, no. 1, pp. 265–268, January 2004.

[21] G. J. Tortora and S. R. Grabowski, "Principles of Anatomy and Physiology", 7th ed. John Wiley and Sons Inc., 1993.

[22] J. Pope, "Medical physics", 2nd ed. Heinemann Educational, 1975.

[23] John Moran Eye Center, University of Utah, H. Kolb, E. Fernandez, and R. Nelson, "Webvision. the organization of the retina and visual system," September 2005, http://webvision.med.utah.edu

[24] H. Kolb, "How the retina works," *American Scientist*, vol. 91, pp. 28–35, January - February 2003.

[25] J. R. Hetling and M. S. Baig-Silva, "Neural prostheses for vision: Designing a functional interface with retinal neurons," *Neurological Research*, vol. 26, pp. 21–34, January 2004.

[26] K. W. Horch and G. Dhillon, "Neuroprosthetics: Theory and Practice", ser. Series on Bioendineering and Biomedical Engineering. World Scientific Publishing Company, April 2004, vol. 2.

[27] B. A. Wandell, A. ElGamal, and B. Girod, "Common principles of image aquisition systems and biological vision," *Proceedings of the IEEE*, vol. 90, no. 1, pp. 5–17, January 2002.

[28] J. Geary, "The Body Electric, an Anatomy of the New Bionic Senses". Weidenfeld and Nicolson, 2002.

[29] M. S. Humayun et Al., "Quantitative study of high dynamic range image sensor architectures," *Survey of Ophthalmology*, vol. 47, no. 4, pp. 335–356, July-August 2002.

[30] Optobionics Corporation, "Optobionics, technology for vision," website, 2006. [Online]. Available: http://www.optobionics.com/

[31] R. A. Normann, E. M. Maynard, K. S. Guillory, and D. J. Warren, "Cortical imaplants for the blind," *IEEE Spectrum*, vol. 33, no. 5, pp. 54–59, May 1996.

[32] K. D. Wise, D. J. Anderson, J. F. Hetke, D. R. Kipke, and K. Najaki, "Wireless implantable microsystems: High-density electronic interfaces to the nervous system," *Proceedings of the IEEE*, vol. 92, no. 1, pp. 76–97, January 2004.

[33] A. E. Gamal and H. Eltoukhy, "CMOS image sensors," *IEEE Circuits and Devices Magazine*, pp. 6–20, May-June 2005.

[34] D. Litwiller, "CCD vs. CMOS: Facts and fiction," *Photonics Spectra*, January 2001.

[35] H. Titus, "Imaging sensors that capture your attention," *Sensor: The journal of Applied Sensing and Technology*, vol. 18, no. 2, February 2001.

[36] J. R. Parker, "Algorithms for Image Processing and Computer Vision", John Wiley and Sons Inc., 1997.

[37] E. R. Davies, "Machine Vision: Theory, Algorithms, Practicalities", 3rd ed. Morgan Kaufmann, December 2004.

[38] J. A. Bangham and S. Marshal, "Image and signal processing with mathematical morphology," *Electronics and Communications Engineering Journal*, vol. 10, pp. 117–128, June 1998.

[39] J. S. J. Lee, R. M. Haralick, and L. G. Shapiro, "Morphological edge detection," *IEEE Journal of Robotics and Automation*, vol. RA-3, no. 2, pp. 142–156, April 1987.

[40] S. Zhang and M. A. Karim, "Euclidean distance transform by stacking filters," *IEEE Signal Processing Letters*, vol. 6, no. 10, pp. 253–256, October 1999.

[41] F. Y.-C. Shih and O. R. Mitchell., "Threshold decomposition of gray-scale morphology into binary morphology," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 1, pp. 31–42, January 1989.

[42] J. C. Thomason, "Black and white gallery," August 2005, http://www.staff.ncl.ac.uk/j.c.thomason

[43] D. M. Reiser, "Sunshine creative," http://www.sunshinecreative.com/

[44] P. Johnson, "Paul johnson photography," 2005, http://www.pauljohnsonphotography.com/

[45] D. J. Smith, "HDL Chip Design", Doon Publications, 1999.

[46] T. N. Swe and K. S. Yeo, "An accurate photodiode model for DC and high frequency spice circuit simulation," *Nanotech: Technical Proceedings of the 2001 International Conference on Modeling and Simulation of Microsystems*, vol. 1, pp. 362–365, 2001.

[47] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, "Digital Integrated Circuits. A Design Perspective", 2nd ed., ser. Electronics and VLSI. Prentice Hall, 2003.

[48] D. H. Neil H.E. Weste, "CMOS VLSI Design. A Circuits and Systems Perspective", 3rd ed. Addison Wesley, 2005.

[49] T. A. DeMassa and Z. Ciccone, "Digital Integrated Circuits". John Wiley and Sons Inc., 1996.

[50] B. Koren, "Photodiodes," *Spie's OE Magazine*, pp. 34–36, August 2001.