Fall 12-2011

A 2-D PROCESSOR ARRAY FOR MASSIVELY PARALLEL IMAGE PROCESSING

Anantha Krishna Nelliparthi
University of Nebraska-Lincoln, krisdare1987@gmail.com

A 2-D PROCESSOR ARRAY FOR MASSIVELY PARALLEL IMAGE PROCESSING

by

Anantha Krishna Nelliparthi

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Electrical Engineering

Under the Supervision of Professors Sina Balkir and Michael W. Hoffman

Lincoln, Nebraska

December, 2011

A 2-D PROCESSOR ARRAY FOR MASSIVELY PARALLEL IMAGE PROCESSING

Anantha Krishna Nelliparthi, M. S.

University of Nebraska, 2011

Adviser: Sina Balkir and Michael W. Hoffman

The concept of introducing image processing logic within the spatial gaps of an array of photodiodes is the key factor behind the presented work. A two-dimensional massively parallel image processing paradigm based on 8×8 pixel neighborhood digital processors has been designed. A low complexity processor array architecture along with its instruction set has been designed and fully verified on a FPGA platform. Various image processing tests have been run on the FPGA platform to demonstrate the functionality of a design that uses 12 parallel processors. The test results indicate that the architecture is scalable to support high frame rates while allowing for flexible processing due to inherent programmability at a high level. The gate level logic synthesis results of the processor targeting a 0.13 μm CMOS technology indicates a low silicon area complexity, allowing for image sensor integration.

DEDICATION

This thesis is dedicated to my beloved parents Jagdish Nelliparthi and Radha Rani Nelliparthi. A special dedication to my maternal great-grandmother late Srimathi Durgamma Thirumareddy, my paternal great-grandfather late Sri Lakshmiah Naidu Nelliparthi, my maternal grandfather late Sri Appa Rao Thirumareddy, my paternal grandfather Sri Lakshmiah Venkata Rao Nelliparthi and my sister late Kumari Swapna Rani Chirugudu.

ACKNOWLEDGMENTS

Firstly, I would like to express my deep gratitude to my adviser Dr. Sina Balkir and I am indebted for his continuous support, valuable guidance and encouragement throughout my Masters program. Also, I am indebted to my co-adviser Dr. Michael W. Hoffman for his continued guidance, support and invaluable inputs throughout my Thesis work. I would like to thank committee member Dr. Khalid Sayood for providing critical review and recommendations for my thesis.

A special thanks to Mr. Nathan Schemm and Mr. Daniel J. White for their valuable assistance for this work. Also, I would like to acknowledge the support of the faculty and staff of Electrical Engineering Department. Finally, I am grateful to my parents, friends and well-wishers for their love, support, patience and encouragement they provided me with, in achieving my Masters degree.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Historical Background and Overview

Recent advances in CMOS fabrication technology have made CMOS image sensors a viable alternative to CCD sensors in multiple applications. In addition to inherently low-cost of fabrication at high levels of integration density, low-power and high speed operation are primary advantages of CMOS technology. More importantly, deep sub-micron CMOS technologies provide the platform for integrating both sensing and processing on the same die. This advantage is highly desirable for implementing stand-alone and portable imaging systems such as smart cameras that require substantial amounts of processing.

Integration of sensing and processing in the context of CMOS image processing systems can generally be categorized into three levels: Chip level, column level, and pixel level processing. Most of the previous work is related to integrating a processing unit at the chip or column level [1]-[2]. However, recent work based on scaling of CMOS image sensors to 0.18 μm processes and below has demonstrated the feasibility of incorporating a processing element at each pixel or a

neighborhood of pixels [3].

In this thesis work, we present a two-dimensional massively parallel image processing paradigm based on 8×8 pixel neighborhood digital processors. While the ultimate goal is the development of a massively parallel single chip imager with pixel neighborhood processing capabilities, as an essential first step, the present work focuses on the neighborhood level parallel processing side of the system. The choice of the neighborhood size is driven by two different but important considerations:

- the spatial image correlation properties typically encountered in image processing applications [4] and
- the necessary footprint for supporting the functionality of a general digital processor without impacting overall scalability.

To that end, a processor array architecture along with its instruction set has been designed. The system has been modeled completely in HDL and a basic version with twelve processors has been implemented on a FPGA to demonstrate image processing tasks.

1.2 Motivation

The motivations for the proposed approach are twofold. The first is the development of massively parallel image processors which support very high frame rates [5]-[7]. In this way, the solution developed will scale without regard to overall imager size and resolution. The second motivation is provided by the fact that the digital processors will allow highly flexible processing choices with random access to different functions and real-time changes in imager functions. While

the parallel neighborhood processors (NPs) will execute in lock step, the individualized neighborhood processor enable capability present in the design will also support region-of-interest processing for selected neighborhoods and reduce power consumption.

1.3 Oraganization of Thesis

The later part of the thesis contains four chapters and the contents of each chapter are structured as follows:

- **Chapter 2** discusses the architecture and instruction set design of the system.
- **Chapter 3** presents the HDL modeling of the system and its prototype FPGA implementation.
- **Chapter 4** presents the FPGA based image processing tests that demonstrate the functionality of the neighborhood processing paradigm.
- **Chapter 5** concludes the report and talks about future prospects.

Chapter 2

Architecture and Design

2.1 Top Level Design

An overall hierarchical view of the design is displayed in figure 2.1. Program Memory, Global Control Unit, Instruction Register and Timing Control Unit are referred to as global modules, shared among all the Neighborhood Processors (NPs).

NP: The term NP refers to an 8×8 pixel neighborhood along with local processing architecture. Each NP runs the program that has been loaded in Program Memory module, simultaneously and independently. The dashed two-sided arrow marks shown in figure 2.1 as a background behind NPs denote possible inter NP communications which will be discussed further in detail. NPs in a particular column share an 8-bit Data Out Bus and a 1-bit Data Out Valid Line. Hence Data can be readout row-wise (row-wise with respect to NPs) along these column shared buses. Program stored into the Program Memory module must ensure that only one NP has access to the column shared Data Out Bus at any given time.

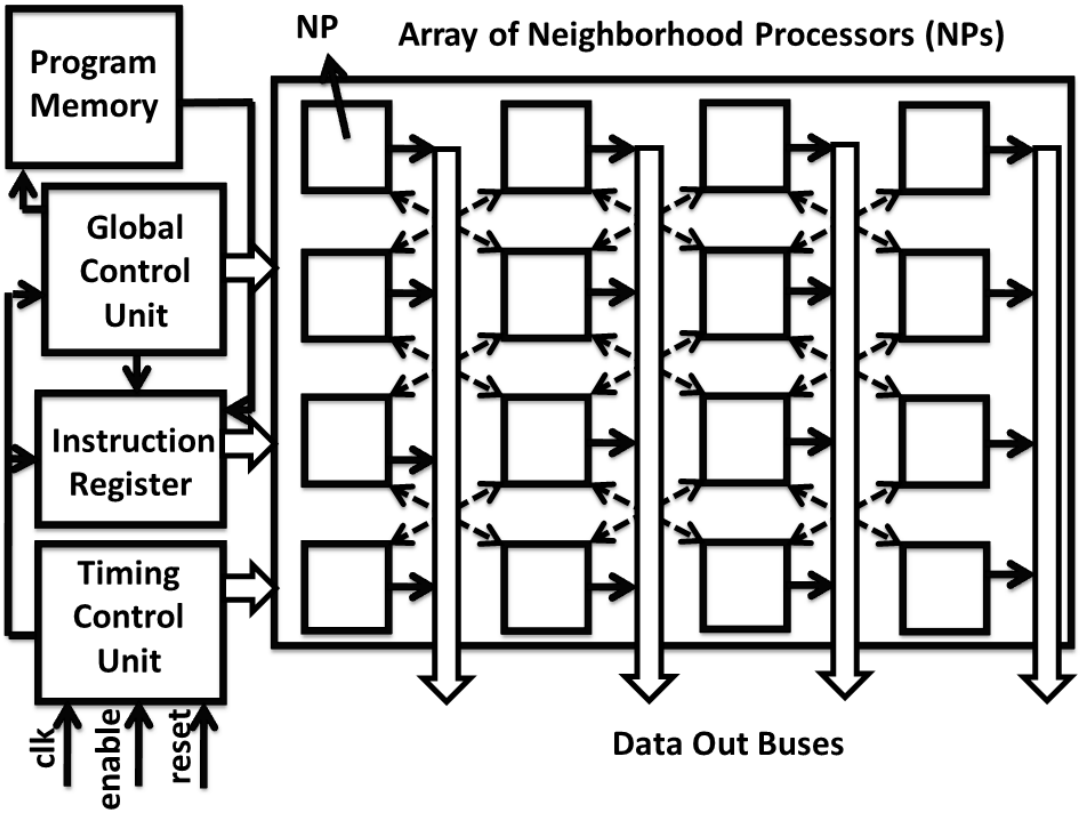

Figure 2.1: Array of neighborhood processors controlled by global modules.

Program Memory: The set of instructions to be performed in sequence are stored in the Program Memory module. The design also facilitates loading of multiple programs into the program memory and calling for the execution of these programs using their starting address in the program memory.

Global Control Unit: The Global Control Unit (GCU) takes care of decoding and interpreting the instructions and accordingly provide control signals. It handles the Program Memory pointer as well as timing for the Instruction Register Latch and takes care of looping while executing jump instructions.

Instruction Register: The Global Control Unit decides which instruction stored in the program memory module has to be executed and accordingly updates the

Instruction Register which is globally shared among all the NPs. Then each NP, according to its current state and GCU control lines, decides whether or not the instruction is to be executed.

Timing Control Unit: The Timing Control Unit handles resetting and initializing the entire system and also provides different control signals and clocks for various modules. Timing waveforms with respect to three crucial clock signals generated by Timing Control Unit are as shown in figure 2.2. These three clock signals are vital for maintaining timing synchronization between the Global Control Unit, the Instruction Register and all the NPs.

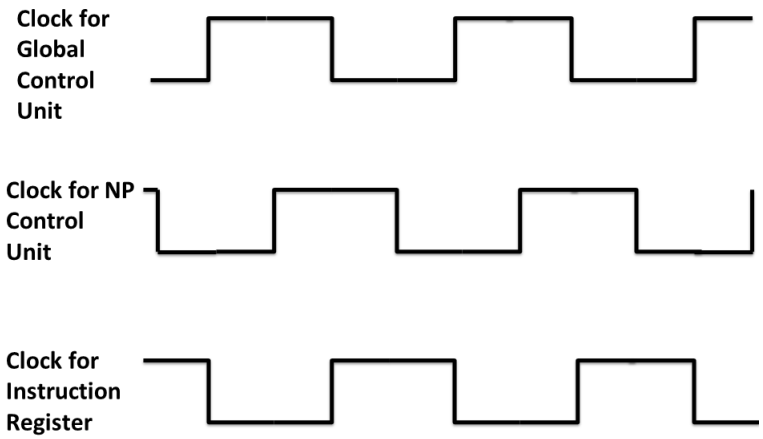

Figure 2.2: Timing Diagram for three clock signals.

2.2 Neighborhood Processor (NP) Interconnectivity

Inter connectivity between multiple NPs is made feasible by introducing shared Neighborhood Registers denoted as NR in figure 2.3. To realize the interconnectivity, we place an array of NPs in space. Neighborhood registers are placed at

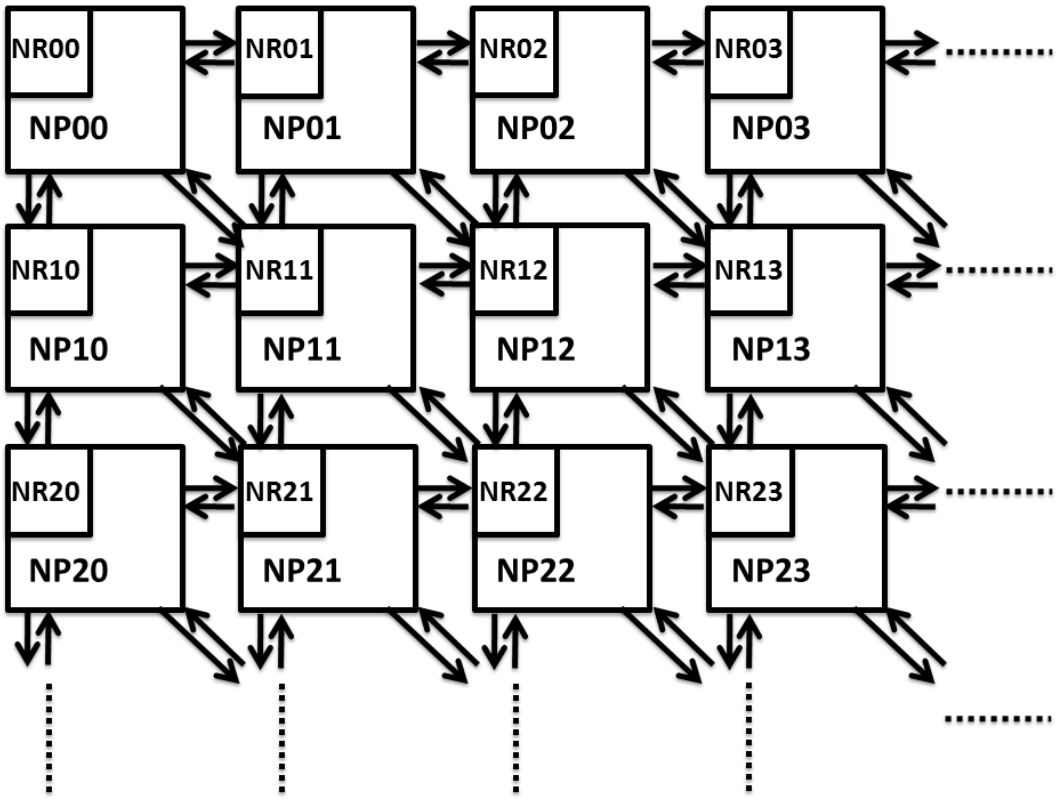

Figure 2.3: Interconnectivity between multiple NPs through NRs.

every junction for communication between NPs. To reduce the complexity, each NP contains a neighborhood register at its top left corner as a part of its internal architecture. Consider NP11 and NR11. NP11 communicates directly with NR11, NR12, NR21 and NR22. NR11 handles all data transfers between NP00, NP01, NP10 and NP11. Similarly, NR12 handles data transfers between NP01, NP02, NP11 and NP12. This allows data transfer in horizontal, vertical, and diagonal directions in just two steps - NP to NR and NR to NP. The significance of this inter data transfer method is apparent when all NPs are functioning in parallel. For example, data can be transferred from NP00 to NP11 via NR11, NP11 to NP22 via NR22, NP22 to NP33 via NR33, and so on, all at once, in two steps.

2.3 Neighborhood Processor (NP) Architecture

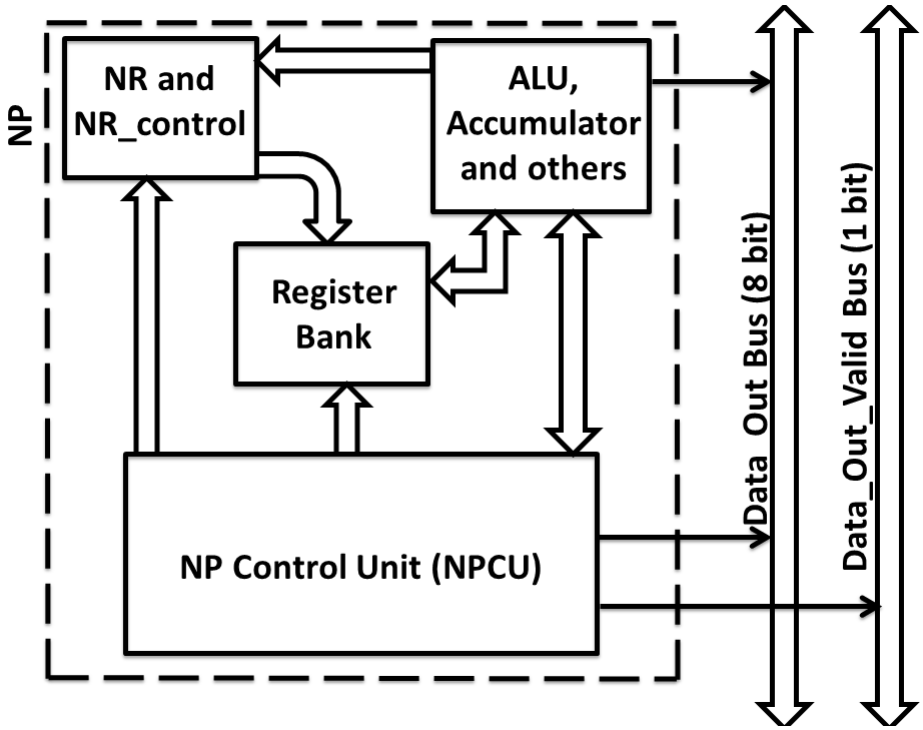

Figure 2.4: NP Architecture

Figure 2.4 illustrates the NP architecture. Inter NP connections, Global Control signals and other globally shared inputs are not shown in figure 2.4, except for column shared buses Data Out and Data Out Valid. The main components of the NP architecture are as follows:

- **Register Bank:** Each 8×8 pixel neighborhood is considered to be a part of the Register Bank, where each pixel location is represented by three 8-bit Pixel Value Registers A, B and C. The architecture is designed such that it provides an option to use register C as two nibbles CH and CL, thereby giving a provision to handle 12-bit data processing. NPs can be programmed to operate in parallel or NPs of a particular ROW or NPs of a particular COLUMN or a single specific NP can be programmed to operate or programmed to stay

inactive while others perform desired tasks, which is achieved by making each NP unique by introducing a Row Column Register inside the register bank of each NP. This register contains a specific unique value with respect to each NP, which can be made use of with the help of the Instruction Set to activate or deactivate a single NP or a group of NPs. NP activation and deactivation is controlled by a single bit inside the Status Register of the NP located inside NP Control Unit. A detailed explanation of Status Register bits is given in Appendix A.

- **NP Control Unit:** The NP Control Unit is the working brain for each NP, as the Global Control Unit is for the entire system. Based on Global Control signals, information in the Instruction Register and information in the Status Register, the NP Control Unit generates control signals for the NP. It also controls the Column Shared Data Out Valid Line.
- **Neighborhood Register and control:** On the top left corner of the NP in figure 2.4, the Neighborhood Register (NR) and its control unit (NR control) can be observed. The NR Control takes control and data inputs from all other neighboring processors and controls the Write operation on NR.
- **ALU, Accumulators, and others:** In addition to an Arithmetic and Logic Unit (ALU), other components of the NP include multiplexers for routing and two 8-bit accumulators ACCA and ACCB. Routing of data inside the NP also allows us to access information in ACCA, ACCB, the row column register, and the status register for read operations using direct register addressing. It also helps the NP Control Unit to perform write operation on the Status Register with assistance from the Instruction Set.

2.4 Instruction Set

Certain features like looping, conditional and unconditional jumps, indirect register addressing and nibble based operations are achieved with the combined effort of the NP Architecture and the Instruction Set Architecture. Our design operates on 8-bit data using a 16-bit instruction set. The entire instruction set can be categorized into five different types as shown in Table 2.1. All five instruction types have the first seven bits common; two bits for condition and five bits for opcode. The first two condition bits involve one unconditional and three conditional executable options for any instruction. The next five bits define either operation or the type of instruction.

Table 2.1: Types of Instruction

Type no.	First 2-bits	Next 5-bits	Remaining 9-bits
I	Condition	Opcode	Address/Immediate
II	Condition	Opcode	Opn and SR Update
III	Condition	Opcode	Jump Toggle and PC-Offset
IV	Condition	Opcode	Opn and Control bits
V	Condition	Opcode	Function bits

- **Type I:** These are Single Operand Instructions involving arithmetic and logical operations excluding shift operations, performed using either of the two 8-bit accumulators.
- **Type II:** Type II operations are Zero Operand Instructions involving all shift operations. Operation (Opn) and SR Update bits define respective shift operation and an option to control Status Register Update through instruction.
- **Type III:** These are Jump Instructions which involve four conditional options

from two Condition bits together with a Jump Toggle bit giving us in total eight different types of jump in a program. PC-Offset refers to the address to be loaded into Program Counter (PC) based on jump condition.

- **Type IV:** One of the interesting features of our instruction set architecture is the Indirect Register Addressing Mode. In this mode of operation we make use of Accumulator B as a register pointer by loading it with the respective register address. Since Accumulator B contains the address, all operations are performed in Accumulator A. Furthermore, the address in Accumulator B can be increased, decreased and modified with respect to any arithmetic and logical operation as feasible through single or zero operand instruction type. The Opn and Control bits define operation and an option to control Status Register Update through instruction.
- **Type V:** Type V instructions are Special Instructions that involve dedicated control functions like resetting the system at instruction level, forcing all NPs to active state, Status Register readout, Nibble mode operation enable and disable, clearing the accumulators, outputting data in accumulators, no operation instruction and end of program.

The complete set of instructions is given in Appendix C.

Chapter 3

Hardware Modeling and Implementation

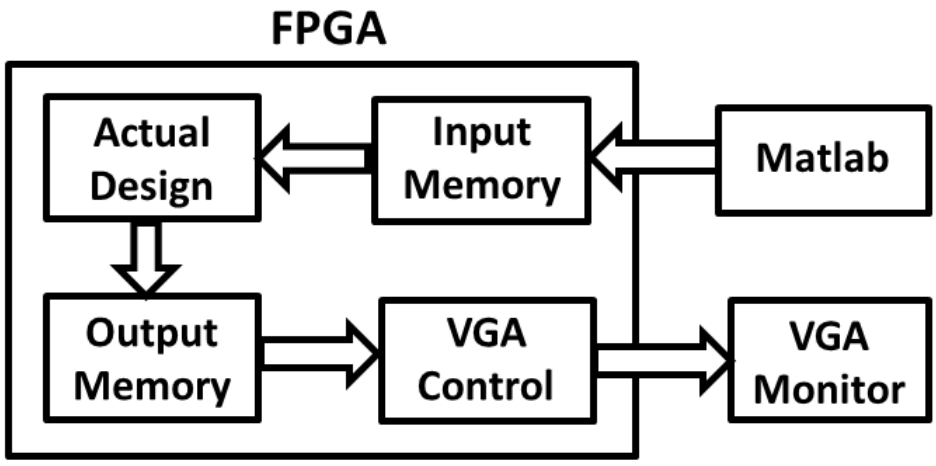

Figure 3.1: FPGA hardware platform for functional testing.

A FPGA based hardware modeling platform has been developed for functional testing of the processing architecture. The entire setup and modeling path is reflected in figure 3.1. First, a binary equivalent of a test image is created via Matlab and is converted into a Verilog RAM representation that acts as the input. The

actual design encompasses the NP array, Global Control Unit, and all the other hardware resources of the processing architecture that include the instructions stored in the Program Memory for executing a series of image processing tasks. The output memory and VGA control sections manage the display of the output image on a VGA monitor.

For hardware test purposes, a 12-NP model has been implemented using a Virtex-5 OpenSPARC Evaluation Platform utilizing the Xilinx Virtex - 5 XC5VLX110T FPGA. Twelve NPs correspond to 768 pixels in a 24×32 (height × width) pixel array. Hence, an input test image has been made of the same size, converted into Verilog code to act as input RAM. The entire code for the actual design and its supporting modules have been represented in Verilog and VHDL. Fidelity of this code has been initially checked for behavioral modeling using the ModelSim environment. Following this step, the design has been synthesized using the Xilinx tools. Next, the ModelSim equivalent for Post-Place and Route Model of the design from the Xilinx tools has been simulated to check post-implementation fidelity. Finally the code is written onto FPGA and the hardware test results that demonstrate certain image processing tasks have been observed on a VGA monitor.

Chapter 4

Test Results and Discussions

4.1 Image Processing Tests

Figure 4.1: Original input test image for 12-NP Model Testing

A series of hardware tests involving programs representing different image processing tasks have been performed on the twelve NP model. Figure 4.1 refers to the actual test image taken as input for a 12-NP model. With regard to output images, one common trait that will be observed is appearance of few undesired lines on the output due to resolution issues with the VGA monitor control. These undesired lines have nothing to do with the output data readout along column

shared buses.

4.1.1 Buffer Test

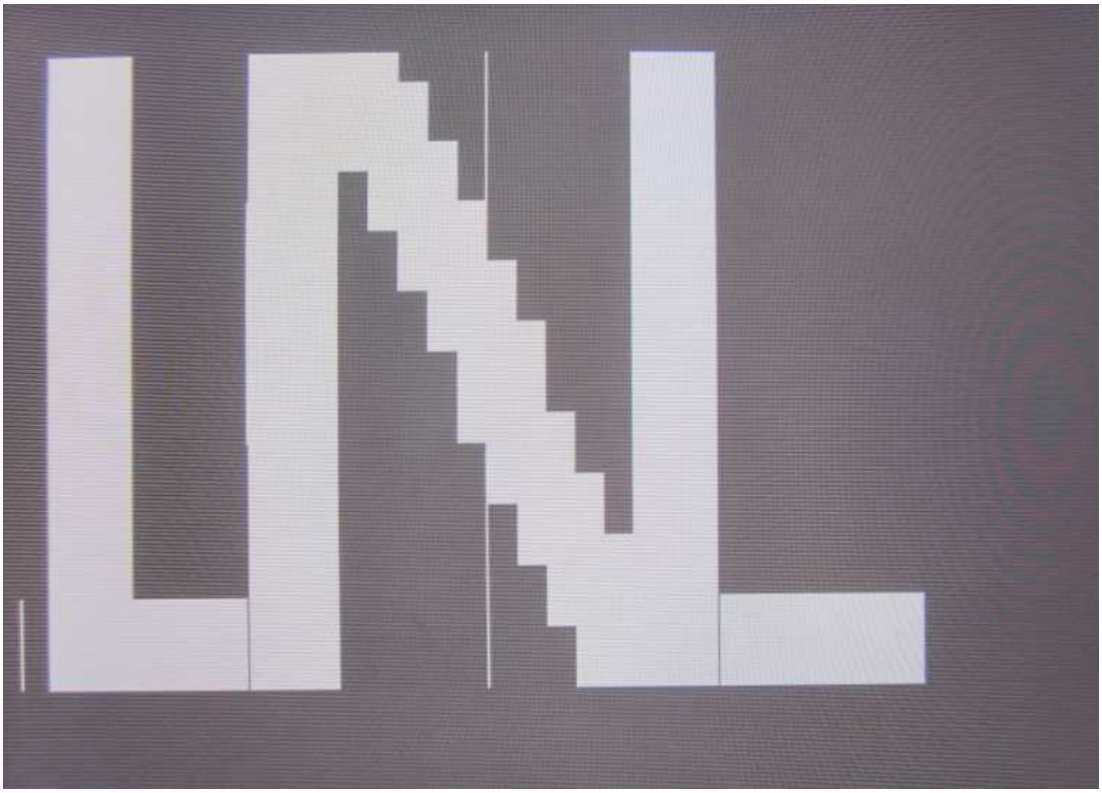

Figure 4.2: Buffered output of the input image

For a clear understanding of the actual pixel-level image of the original image, the input test image has been buffered through the 12-NP Model and the output image has been scaled and observed on a monitor with the help of VGA Control module. Figure 4.2 illustrates the output image as seen on the monitor.

4.1.2 Invert Test

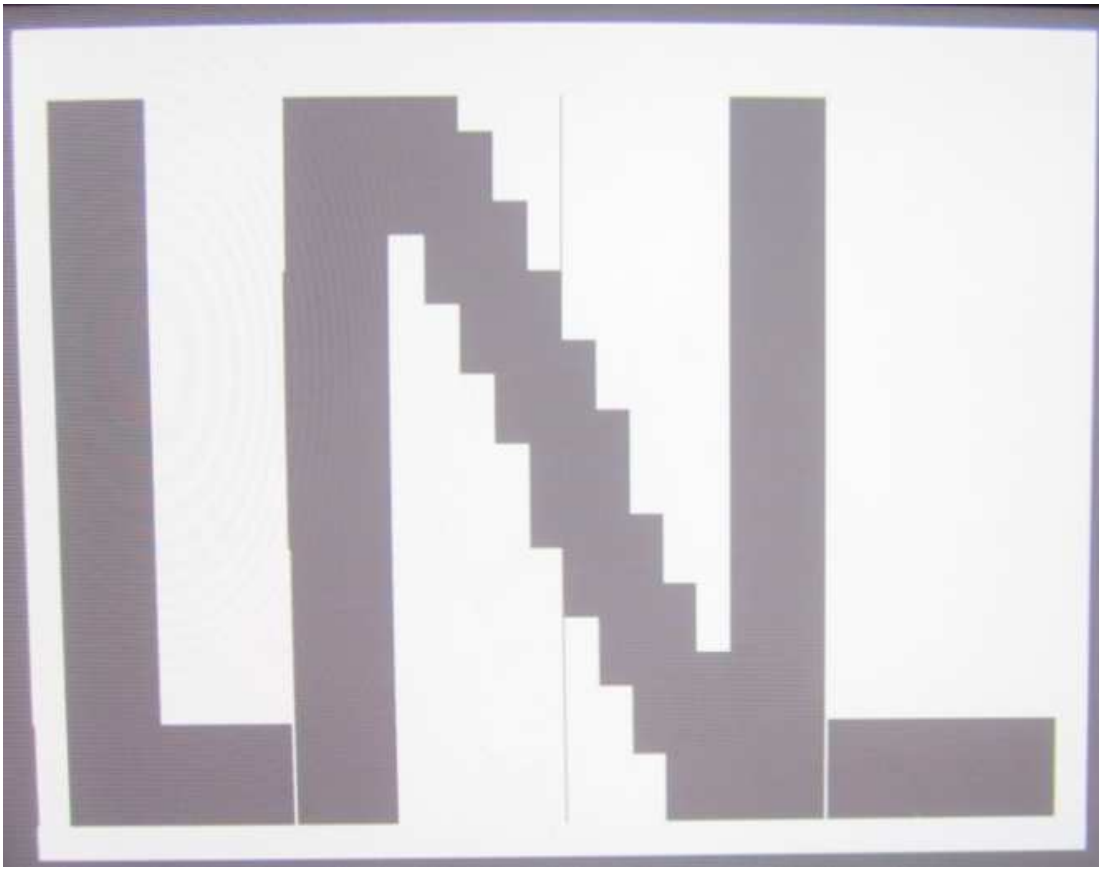

Figure 4.3: Inverted output of the input image

An invert program has been loaded into the Program Memory module and the output on VGA Monitor is illustrated in figure 4.3.

4.1.3 Right Shift Test

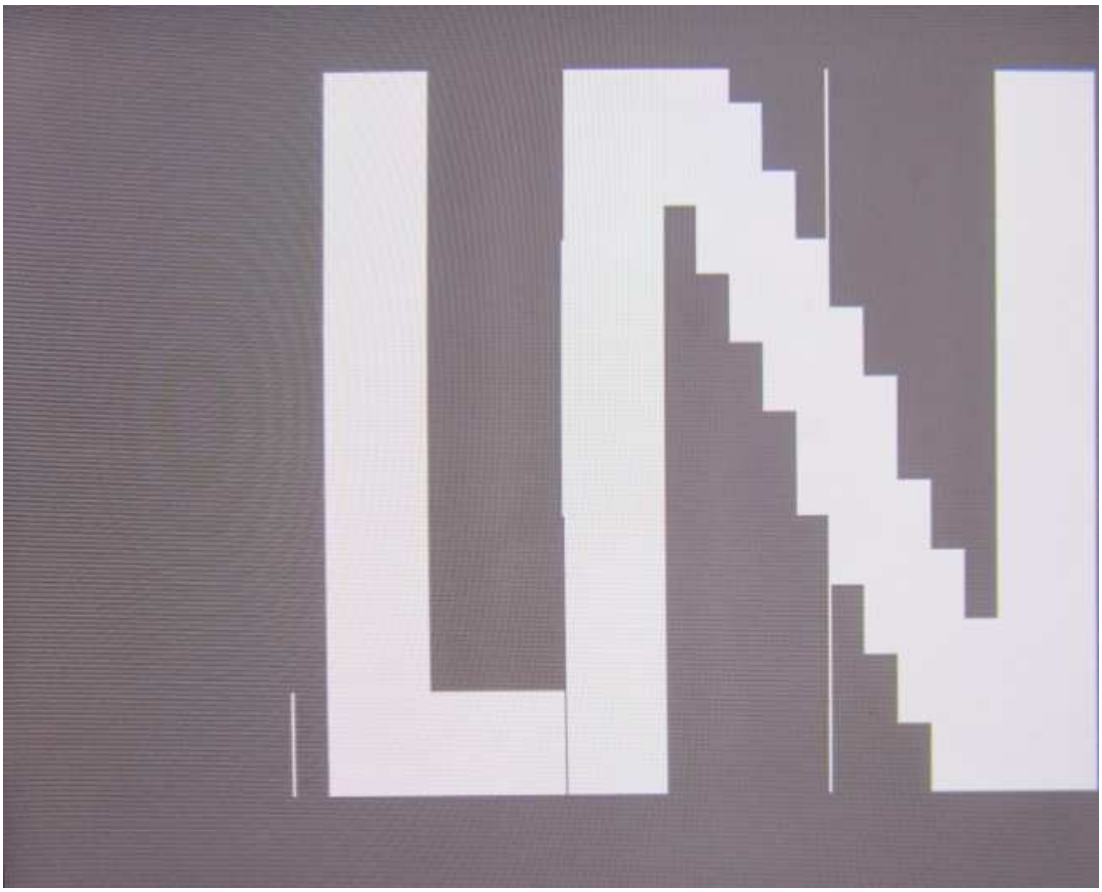

Figure 4.4: Right shift of the input image

All NPs work in parallel and making use of this advantage, we can shift an entire image by involving a program that basically moves data from one NP to another through Neighborhood Registers. Figure 4.4 reflects an image shift equivalent to one NP in right direction.

4.1.4 Top Right Shift Test

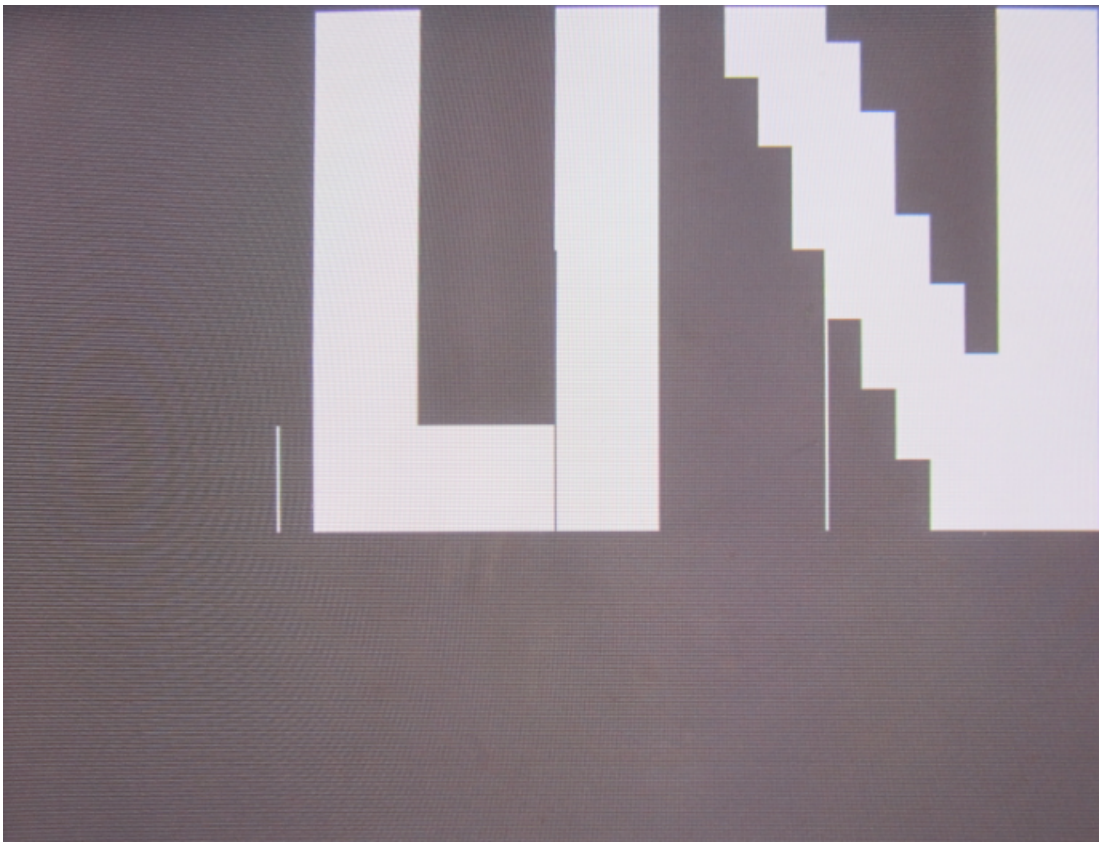

Figure 4.5: Top Right Shift of the input image

Figure 4.5 reflects the output image when a diagonal Top Right Shift of the input image is performed. Due to the distributed sharing of Neighborhood Registers among adjacent NPs, this diagonal shift of image involves the same time as it takes for Right Shift of the input image.

4.1.5 Horizontal Edge Detection Test

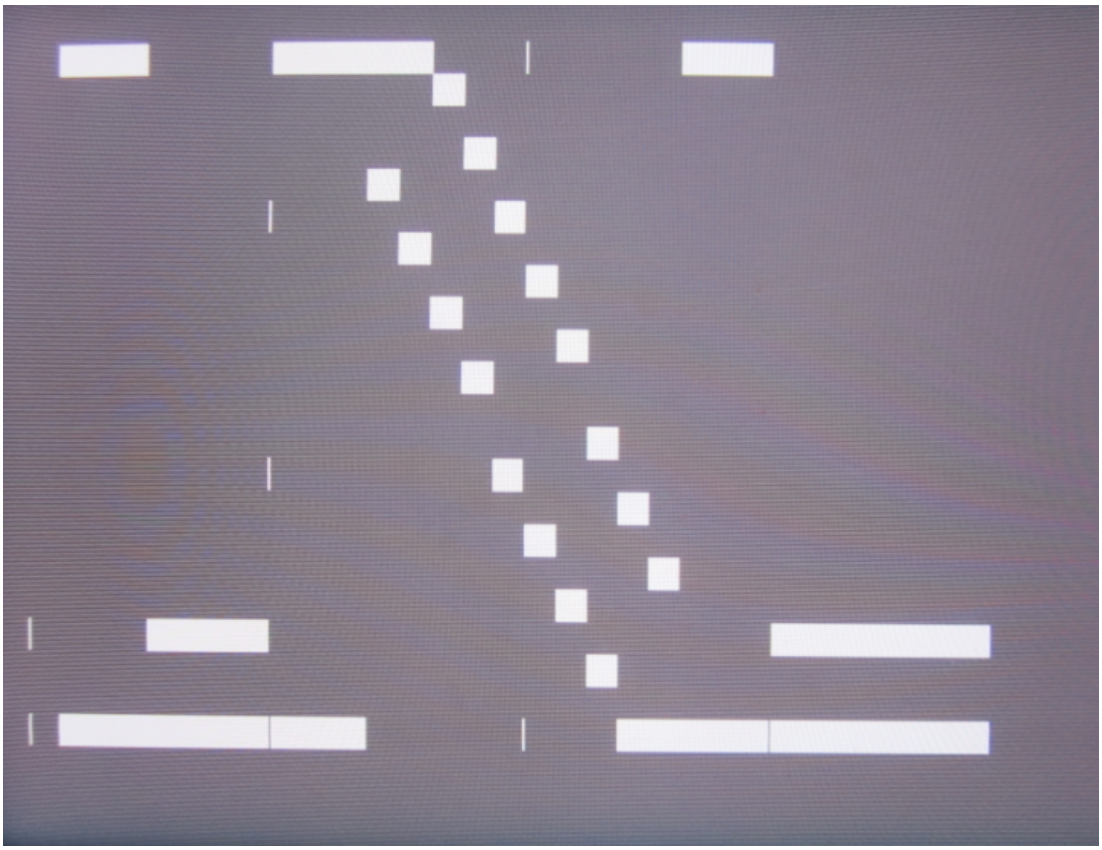

Figure 4.6: Horizontal Edge Detection on the input image

The input image is scanned vertically pixel-wise NP-parallel for detecting horizontal edges and the output is as illustrated in figure 4.6. It involved transfer of data between NPs through Neighborhood Registers.

4.1.6 Vertical Edge Detection Test

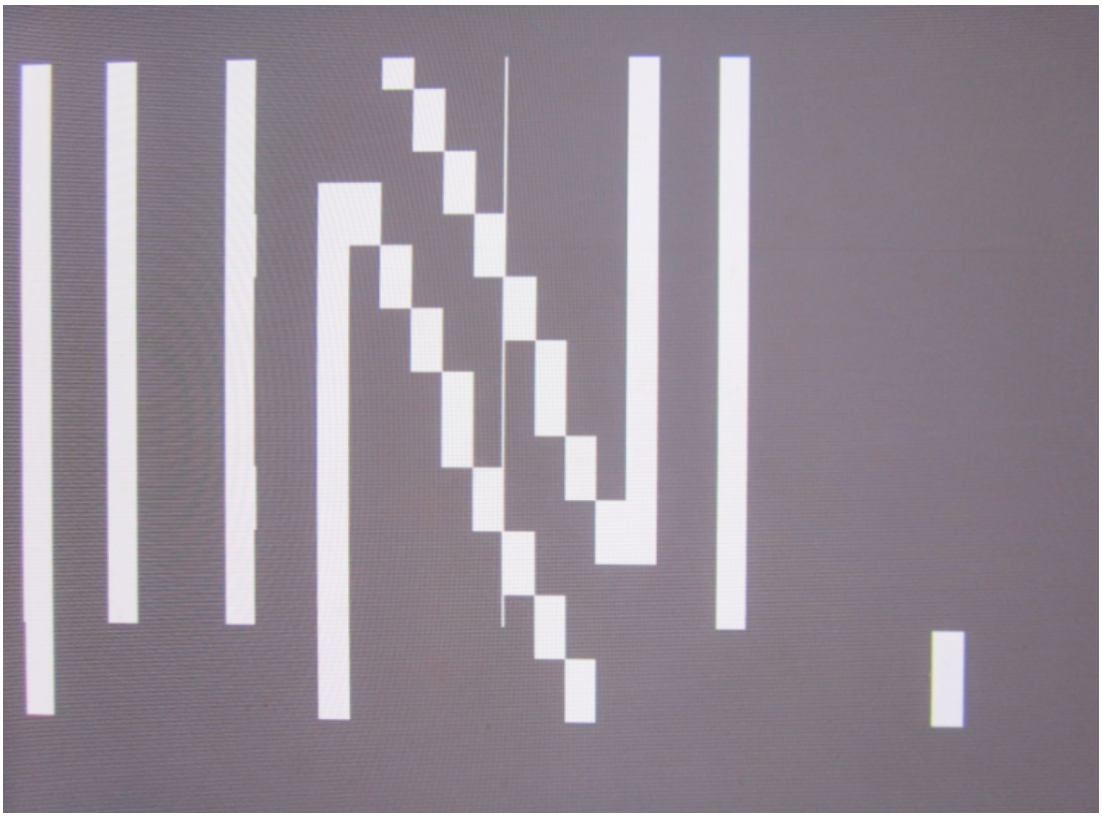

Figure 4.7: Vertical Edge Detection on the input image

The input image is scanned horizontally pixel-wise NP-parallel for detecting vertical edges and the output is as illustrated in figure 4.7. It involved transfer of data between NPs through Neighborhood Registers.

4.1.7 Total Edge Detection Test

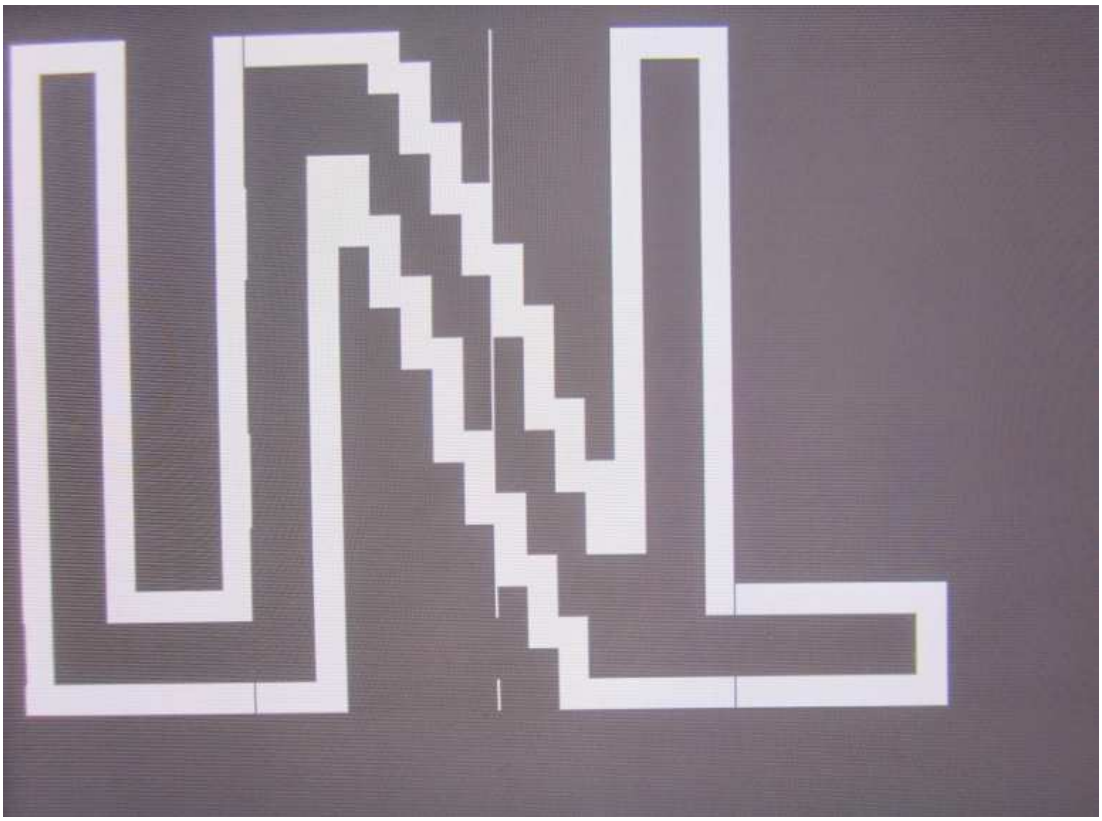

Figure 4.8: Total Edge Detection on the input image

The input image is scanned diagonally (three ways - right, bottom and right-bottom) pixel-wise NP-parallel for detecting its edges and the output is as illustrated in figure 4.8. It involved transfer of data between NPs through Neighborhood Registers.

4.2 Speed Comparison

Tests between 2-NP (1×2 NP-wise or 8×16 pixel-wise) and 12-NP (3×4 NP-wise or 24×32 pixel-wise) are performed to show the practical effect scalability has on parallel processing and parallel readout. ModelSim simulation environment has been employed to look into the number of clock cycles taken for each test by 2-NP and 12-NP, which are as shown in Table 4.1

Table 4.1: No. of clock cycles for 2-NP and 12-NP model Tests

Test	2-NP	12-NP
Invert	837	837
Total Edge Detection	4337	4337
Readout	731	2193

For the Invert image and Edge Detection tests, since the pixels are processed by their respective neighborhood processors, the time taken for 2-NP and 12-NP models is the same. For the Readout test, since 12-NP model has 3 rows of NPs and 2-NP model has only one row, time taken for 12-NP model readout is three times the time taken for 2-NP model.

4.3 Gate Level Hardware Complexity

The ultimate goal of the presented work is to integrate each NP with CMOS image sensors on a focal plane that houses an 8×8 pixel neighborhood. To that end, an assessment of gate level hardware complexity of the NP has been performed with a target 0.13 μm CMOS implementation technology. Table 4.2 displays the breakdown of gate level components obtained from the logic synthesis of a single

NP executed in the Cadence Design Environment and configured for a 0.13 μm commercial grade CMOS standard cell library.

Table 4.2: Gate level logic synthesis results for a single NP

Type	Instances	Area %
sequential	1606	57.6
inverter	618	3.2
tristate	537	7.3
logic	3012	31.9

Based on the individual area utilization of the gates and the regularity of the sequential components in terms of layout placement, the presented NP architecture's gate level complexity allows for the future development of a first generation massively parallel single chip imager with pixel neighborhood processing.

4.4 Scalability Considerations

One of the key motivators for the described architecture is scalability. The fact that a single NP controls an 8×8 pixel neighborhood implies we can expand our design by factors of 8, row-wise or column-wise. Since all the NPs process in parallel and since the NP circuitry will be placed within the spatial arrangement of pixel neighborhood, scaling of total pixel neighborhood does not introduce extra logical complexity upto the Global Control Unit (GCU) limit, which is 16×16 NPs giving rise to 128×128 grid of pixels. Beyond that the only extra logical addition is to introduce another Global Control Unit, which consumes relatively little area in comparison with 128×128 grid of pixels. With respect to readout or Input/Output constraints, each column refers to an 8-bit Data Out Bus. Hence, for a 128×128

grid of pixels, we need only 16 8-bit Data Out Buses. Mathematically speaking the relation between number of rows, columns, NPs and neighborhood can be seen in Table 4.3.

Table 4.3: Mathematical understanding on scalability.

NPs	pixels	GCU	8-bit buses
$r \times c$ *	$8r \times 8c$	$\max(\lceil r/16 \rceil, \lceil c/16 \rceil)$	c
16×16	128×128	1	16
32×32	256×256	4	32
1×2	8×16	1	2
3×4	24×32	1	4

*r: rows, c: columns, $\lceil x \rceil$ is least integer $\geq x$

Chapter 5

Conclusions and Future Prospects

5.1 Conclusions

A two-dimensional massively parallel image processing paradigm based on 8×8 neighborhood digital processors has been presented. The ultimate goal is the development of a massively parallel single chip imager with pixel neighborhood processing capabilities. As the initial step, the presented work focuses on the neighborhood level parallel processing side of the system. To that end, a low complexity processor array architecture along with its instruction set has been designed and fully verified on an FPGA platform.

5.2 Future Prospects

5.2.1 Massively parallel single chip CMOS imager

One of the motivating factors for this design is to reduce rise in extra complexity (complexity beyond neighborhood processor per 8×8 pixel array) when the de-

sign is scaled to higher resolution. Section 4.3 and Section 4.4 reflect the feasibility of this goal. Hence, there is a great opportunity for future work which will focus on the development of a first generation massively parallel single chip CMOS imager that incorporates the presented pixel neighborhood processing techniques.

5.2.2 Expansions in instruction set

Nine opcodes are available for expansion in the instruction set. Further, there is plenty of scope for expansion in instructions of Type V. Hence, it reflects another prospect for future work involving addition of more functions or commands to the existing instruction set and modifications in NP architecture.

5.2.3 Address Event Representation (AER)

A video output involves transfer of numerous frames, where each frame represents a static image at a particular instant. Higher resolutions storage and transfer of each individual frame information involves huge memory modules and high power consumption for data transfer. Address Event Representation reduces the size of the data to be transferred and thus saves storage space and power consumption involved in data transfer.

Initially, the very first frame of the video is transmitted. Then in the array of pixels taking multiple frames, each pixel compares its previous recorded value with the new one and generates an interrupt if the difference crosses a certain threshold. The time of interrupt generation and the address of the pixel that generated the interrupt are saved in an interrupt memory. Then one after another these interrupts are processed and the time of interrupt generation, the address of the pixel that generated the interrupt and the difference of frame values are transmitted. Thus,

information is passed only if a significant change is observed in frames and only information with respect to the pixels involved with the change are transmitted thereby saving storage space and power consumption. This kind of data representation is termed as Address Event Representation. With few modifications and additions to the existing design AER can be realized.

5.2.4 Low power/Reign of interest

The ability to turn on or turn off a specific NP or row of NPs or column of NPs, indicates that the design supports reign of interest and also shows prospects for low power applications.

5.2.5 Unexplored features

Certain features of the design like making use of nibble based operation to perform 12-bit operations and making use of shift operations to perform multiplications are yet to be explored further though they have been included and verified in the design.

5.2.6 Compiler

Currently the design takes machine code out of assembly language using an assembler. This is favorable for initial testing and debugging of the design through instructions. Since the functionality has been successfully verified, a compiler can be introduced to simplify the process of writing program for the presented design.

Appendix A

Status Register

The concept of having only one NP or row of NPs or column of NPs to either perform desired tasks or be prevented from performing certain tasks is achieved with the help of Status Register and a series of instructions involved in a program. All NPs have a Status Register in their NP Control Unit. A particular bit in this Status Register is responsible for activating or deactivating a NP. Table A.1 refers to 8 bits of the Status Register.

Table A.1: Status Register bits.

MSB7	MSB6	MSB5	MSB4	MSB3	MSB2	MSB1	MSB0
Free bit	NP ON	SRU	U	Z	N	O	C

- **C:** The Carry bit is set whenever carry is observed from output of ALU involving an addition operation or whenever borrow is observed from output of ALU involving a subtraction operation.
- **O:** The Overflow bit is set whenever addition of two positive numbers gives out a negative number at ALU output.

- **N:** The Negative bit is set whenever a negative number is observed at the output of ALU.
- **Z:** The Zero bit is set whenever ALU output is observed to be of zero value.
- **U:** The Underflow bit is set whenever addition of two negative numbers gives out a positive number at ALU output.
- **SRU:** The Status Register Update bit is that bit which defines whether or not the above 5 bits; Carry, Overflow, Negative, Zero and Underflow are to be updated or not. If SRU is high, these 5 bits will be updated at each instruction accordingly else they remain in their previous state. SRU can be turned ON or OFF with a series of instructions which finally end with performing a write operation on Status Register such that only SRU bit is altered while others remain the same. Type II and Type IV instructions can momentarily (only for that instruction) allow update on these 5 bits even if SRU bit is in low state.
- **NP ON:** This bit defines whether or not a NP participates in executing the instructions. If this bit is high, then NP executes the set of instructions (provided condition bits must be satisfied for that specific instruction). If this bit is low, then NP remains in a low state. This bit goes high when the system is reset using the reset instruction. It can be brought low by making use of a series of instructions that involve the Row Column Register of the NP. Once it is turned low, NP becomes inactive to all instructions except for Type V reset instruction or Type V instruction that turns all NPs to ON state.
- **Free bit:** This bit is available for further expansions and currently has no significance.

Appendix B

Register Addressing

The Current Design makes use of 16-bit instruction of which lower 9 bits are generally used for defining immediate value or for defining register address. Each NP has a Register Bank that comprises of 64×3 pixel registers along with 8 general registers like Row Column Register (Register that contains a unique value for each NP), Neighborhood Register, etc. Table B.1 refers to addressing mode when Nibble based operations are disabled, implying that pixel register C is used as a complete 8-bit register.

Table B.2 refers to addressing mode when Nibble based operations are enabled, implying that pixel register C is utilized as two Nibbles CL and CH.

The Assembler realization of Immediate value or Register addressing is explained in Table B.3

Table B.1: Addressing when Nibble mode Off.

Reg/Imm	Address/Value	Example
Imm value	1 and any imm data	1 10001110
Pixel reg A	0 00 and then column number, row number of pixel	0 00 010 001 ('A' register of pixel addressed at 3rd column and 2nd row)
Pixel reg B	0 01 and then column number, row number of pixel	0 01 000 000 ('B' register of pixel addressed at 1st column and 1st row)
Pixel reg C	0 10 and then column number, row number of pixel	0 10 001 010 ('B' register of pixel addressed at 2nd column and 3rd row)
Status Register	0 11 000 000	0 11 000 000
ACCA	0 11 000 001	0 11 000 001
ACCB	0 11 000 010	0 11 000 010
Row Column Register	0 11 000 011	0 11 000 011
Top Left NR	0 11 000 100	0 11 000 100
Top Right NR	0 11 000 101	0 11 000 101
Bottom Left NR	0 11 000 110	0 11 000 110
Bottom Right NR	0 11 000 111	0 11 000 111

Table B.2: Addressing when Nibble mode On.

Reg/Imm	Address/Value	Example
Imm value	1 and any imm data	1 10001110
Pixel reg A	0 00 and then column number, row number of pixel	0 00 010 001 ('A' register of pixel addressed at 3rd column and 2nd row)
Pixel reg B	0 01 and then column number, row number of pixel	0 01 000 000 ('B' register of pixel addressed at 1st column and 1st row)
Pixel nibble CL	0 10 and then column number, row number of pixel	0 10 001 010 ('CL' nibble of pixel addressed at 2nd column and 3rd row)
Pixel nibble CH	0 11 and then column number, row number of pixel	0 11 001 010 ('CH' nibble of pixel addressed at 2nd column and 3rd row)

Table B.3: Assembler realization of register addressing or immediate value.

Assembler Language	Interpretation
I AF	Immediate value = hexadecimal AF
A A 01	Address;Reg A;Pixel 1st column 2nd row
A B 10	Address;Reg B;Pixel 2nd column 1st row
A C 00	Address;Reg C;Pixel 1st column 1st row
A CL 00	Address;Nibble CL;Pixel 1st column 1st row
A CH 00	Address;Nibble CH;Pixel 1st column 1st row
A S 00	Address;Status Register
A S 01	Address;ACCA
A S 02	Address;ACCB
A S 03	Address;Row Column Register
A S 04	Address;Top Left NR
A S 05	Address;Top Right NR
A S 06	Address;Bottom Left NR
A S 07	Address;Bottom Right NR

Appendix C

Complete Instruction Set

The first 7 bits for any Instruction denote condition bits and operation code (op-code). The first 2 bits have same significance and relevance to all the 5 types of instruction except for Type III, which has a little different interpretation. The next 5 bits referred to as opcodes are multiple for Type I and II, while its only one unique value for other 3 types. Each Instruction Type is discussed in detail through a series of tables.

C.1 Type I

The first 2 bits of Type I instruction have four options and are realized as explained in Table C.1.

The next 5 bits refer to opcodes which involve Arithmetic and Logic operations as explained in Table C.2. All the operations involve either ACCA or ACCB with either immediate data or register data as referenced by the remaining 9 bits.

Sample instructions for Type I are given in Table C.3

Table C.1: Type I - First 2 bits (15:14).

Assembly Language	Bits	Interpretation
Z	00	Execute instruction if Zero bit is set
C	01	Execute instruction if Carry bit is set
N	10	Execute instruction if Negative bit is set
U	11	Execute instruction unconditionally

Table C.2: Type I - 5 opcode bits (13:9).

Assembly Language	Bits	Interpretation
ADDA	00000	Add data to ACCA
ADDB	10000	Add data to ACCB
ADCA	00001	Add with carry to ACCA
ADCB	10001	Add with carry to ACCB
SUBA	00010	Subtract from ACCA
SUBB	10010	Subtract from ACCB
SBBA	00011	Subtract with borrow from ACCA
SBBB	10011	Subtract with borrow from ACCB
ANDA	00100	LOGICAL AND with ACCA
ANDB	10100	LOGICAL AND with ACCB
ORA	00101	LOGICAL OR with ACCA
ORB	10101	LOGICAL OR with ACCB
XORA	00110	LOGICAL XOR with ACCA
XORB	10110	LOGICAL XOR with ACCB
LOADA	01100	LOAD ACCA with data
LOADB	11100	LOAD ACCB with data
MOVA	01101	MOVE ACCA data to addressed register
MOVB	11101	MOVE ACCB data to addressed register

Table C.3: Type I - instruction examples.

Instruction	Interpretation
U ADDA A A 00	Add to ACCA the data in Register A of pixel at 1st column 1st row, unconditionally.
Z LOADA I AE	LOAD ACCA with immediate value AE(hexadecimal) if Zero bit of Status Register is set.

C.2 Type II

The first 2 bits of Type II instruction have four options and are realized as explained in Table C.4.

Table C.4: Type II - First 2 bits (15:14).

Assembly Language	Bits	Interpretation
Z	00	Execute instruction if Zero bit is set
C	01	Execute instruction if Carry bit is set
N	10	Execute instruction if Negative bit is set
U	11	Execute instruction unconditionally

The next 5 bits refer to only 2 opcodes defining whether the operation is on ACCA or ACCB as explained in Table C.5

Table C.5: Type II - Opcode 5 bits (13:9).

Assembly Language	Bits	Interpretation
A	00111	Shift operation on ACCA
B	10111	Shift operation on ACCB

The next 4 bits (8:5) define the type of Shift operation whether it is right or left etc. as explained in Table C.6.

Table C.6: Type II - Shift operation bits (8:5).

Assembly Language	Bits	Interpretation
ASR	0111	Arithmetic Shift to Right
SR	1000	Shift Right
SRC	1001	Shift Right with Carry
SL	1010	Shift Left
SLC	1011	Shift Left with Carry

Of the next 5 bits (4:0); bit (4) defines whether Status Register bits - Zero, Carry, Negative, Overflow and Underflow are to be updated or not for the respective instruction as shown in Table C.7. Remaining 4 bits are of no significance now and can be used for future developments. By default they are taken to be zero.

Table C.7: Type II - Status Register (SR) Update bits (4:0).

Assembly Language	Bits	Interpretation
0	00000	Do not update SR
1	10000	Update SR

Sample instructions for Type II are given in Table C.8

C.3 Type III

The first 2 bits of Type II instruction have four options which along with the Jump toggle bit (bit 8) give in total eight different jumps as explained in Table C.9.

Table C.8: Type II - instruction examples.

Instruction	Interpretation
Z A SR 0	If Zero bit of Status Register is set, Shift Right the data in ACCA by 1 bit without updating Status Register.
U B SLC 1	Unconditionally, Shift Left the data in ACCB by 1 bit along with carry and update Status Register.

Table C.9: Type III - Jump Conditions (15:14),(8).

Assembly Language	Bits	Interpretation
Z 0	000	Jump if Zero bit is set
Z 1	001	Jump if Zero bit is not set
C 0	010	Jump if Carry bit is set
C 1	011	Jump if Carry bit is not set
N 0	100	Jump if Negative bit is set
N 1	101	Jump if Negative bit is not set
U 0	110	Jump if Overflow bit is set
U 1	111	Unconditional Jump

The opcode for Type III instructions is same and so the five bits (13:9) remain as 11110. The remaining 8 bits (7:0) denoted as PC-Offset refer to the value to be loaded into Program Counter if jump condition is satisfied. Sample instructions for Type III are given in Table C.10

Table C.10: Type III - instruction examples.

Instruction	Interpretation
N JUMP 0 04	If Negative bit of Status Register is set, load Program Counter with hexadecimal value 04.
Z JUMP 1 DF	If Zero bit of Status Register is not set, load Program Counter with hexadecimal value DF.

C.4 Type IV

The first 2 bits of Type IV instruction have four options and are realized as explained in Table C.11.

Table C.11: Type IV - First 2 bits (15:14).

Assembly Language	Bits	Interpretation
Z	00	Execute instruction if Zero bit is set
C	01	Execute instruction if Carry bit is set
N	10	Execute instruction if Negative bit is set
U	11	Execute instruction unconditionally

The next 5 bits (13:9) defining the opcode for Type IV instruction remain same at 01110. Type IV involve all Type I and II operations but with respect to ACCA as the working Accumulator and ACCB as a register address pointer. The next 4 bits define arithmetic, logic and shift operations as shown in Table C.12. In Table C.12, data refers to information in register pointed by the address stored in ACCB.

Of the remaining 5 bits (4:0); bit (4) refers to Status Register Update, bit (3) refers

Table C.12: Type IV - operations bits (8:5).

Assembly Language	Bits	Interpretation
ADD	0000	Add data to ACCA
ADC	0001	Add data with carry to ACCA
SUB	0010	Subtract data from ACCA
SBB	0011	Subtract with borrow from ACCA
AND	0100	LOGICAL AND with ACCA
OR	0101	LOGICAL OR with ACCA
XOR	0110	LOGICAL XOR with ACCA
ASR	0111	Arithmetic Right Shift the value in ACCA
SR	1000	Shift Right the value in ACCA
SRC	1001	Shift Right with Carry the value in ACCA
SL	1010	Shift Left the value in ACCA
SLC	1011	Shift Left with Carry the value in ACCA
MOV	1100	MOVE value in ACCA to address pointed by ACCB
LOAD	1100	LOAD data into ACCA from address pointed by ACCB

to Read (Here Read refers to all operations mentioned in Table C.12 other than MOV operation) or Write operation and remaining 3 bits are free bits for future developments. If we observe Table C.12, LOAD and MOV both denote same operational bit information. Hence, bit (3) is used to denote Read or Write operation (LOAD or MOV respectively). The bit values for bits (4:0) are as shown in Table C.13.

Sample instructions for Type IV are given in Table C.14

Table C.13: Type IV - Status Register (SR) Update and Rd/Wr bits (4:0).

Assembly Language	Bits	Interpretation
0R	00000	Do not update SR. Not a MOV operation.
1R	10000	Update SR. Not a MOV operation.
0W	01000	Do not update SR. MOV operation.

Table C.14: Type IV - instruction examples.

Instruction	Interpretation
N IRAM ADC 1R	If Negative bit of Status Register is set; Add with Carry the data in register addressed by ACCB, to ACCA and update Status Register
Z IRAM MOV 0W	If Zero bit of Status Register is set; Move the data in ACCA to register addressed by ACCB

C.5 Type V

The first 2 bits of Type V instruction have four options and are realized as explained in Table C.15.

Table C.15: Type V - First 2 bits (15:14).

Assembly Language	Bits	Interpretation
Z	00	Execute instruction if Zero bit is set
C	01	Execute instruction if Carry bit is set
N	10	Execute instruction if Negative bit is set
U	11	Execute instruction unconditionally

The next 5 bits (13:9) defining the opcode for Type V instruction remain same at 01111. The remaining bits are realized as displayed in Table C.16.

Table C.16: Type V - Instructions. Bits (8:0).

Assembly Language	Bits	Interpretation
FOPN	100000000	Force all NPs to ON State
RST	110000000	Reset the entire system
SROUT	111000000	Output Status Register on Data Out Bus
NOP	010000000	No Operation/Idle Instruction
END	001000000	End of Program
OUTA	000100000	Output ACCA on Data Out Bus
OUTB	000110000	Output ACCB on Data Out Bus
NBDS	000001000	Nibble Mode Operation Disable
NBEN	000001100	Nibble Mode Operation Enable
CLRA	000000010	Clear Accumulator A
CLRB	000000011	Clear Accumulator B

Sample instructions for Type V are given in Table C.17

Table C.17: Type V - instruction examples.

Instruction	Interpretation
Z SPL OUTA	If Zero bit of Status Register is set; Output ACCA on Data Out Bus
U SPL NBEN	Unconditionally, Enable Nibble Mode Operation

Further, 9 opcodes are available for future expansion of the instruction set.

Bibliography

[1] L. Kozlowski, G. Rossi, L. Blanquart, R. Marchesini, Y. Huang, G. Chow, J. Richardson, and D. Standley, "Pixel noise suppression via SoC management of target reset in a 1920 x 1080 CMOS image sensor," *IEEE J. Solid-State Circuits*, vol. 39, no. 9, pp. 1487-1496, Sep. 2004. 1.1

[2] M. Sakakibara, S. Kawahito, D. Handoko, N. Nakamura, M. Higashi, K. Mabuchi, and H. Sumi, "A high-sensitivity CMOS image sensor with gain-adaptive column amplifiers," *IEEE J. Solid-State Circuits*, vol. 40, no. 5, pp. 1147-1156, May 2005. 1.1

[3] J. Dubois, D. Ginhac, M. Paindavoine, and B. Heyrman, "A 10 000 fps CMOS sensor with massively parallel image processing," *IEEE J. Solid-State Circuits*, vol. 43, no. 3, pp. 706-717, Mar. 2008. 1.1

[4] K. Sayood, *Introduction to Data Compression*, 3rd ed. San Francisco, CA: Elsevier, 2005. 1.1

[5] M. Ishikawa and T. Komuro, "Digital vision chips and high-speed vision systems," in *Symp. VLSI Circuits Dig. Tech. Papers*, 2001, pp. 14. 1.2

[6] L. Lindgren, J. Melander, R. Johansson, and B. Moller, "A multiresolution 100-GOPS 4-Gpixels/s programmable smart vision sensor for multi-sense imaging," *IEEE J. Solid-State Circuits*, vol. 40, no. 6, pp. 13501359, Jun. 2005.

[7] Y. Sugiyama, M. Takumi, H. Toyoda, N. Mukozaka, A. Ihori, T. Kurashina, Y. Nakamura, T. Tonbe, and S. Mizuno, "A high-speed CMOS image with profile data acquiring function," *IEEE J. Solid-State Circuits*, vol. 40, pp. 28162823, Dec. 2005.

1.2