IMPLEMENTATION OF
IMAGE PROCESSING TECHNIQUE IN FPGA

By

Ng Soo Kheng

Report submitted to

UNIVERSITI SAINS MALAYSIA

In partial fulfillment of the requirements for the degree of

Bachelor of Engineering (Mechatronics Engineering)

School of Engineering

Electrical and Electronic

Universiti Sains Malaysia

May 2006

ABSTRAK

IMPLEMENTASI TEKNIK PEMPROSESAN IMEJ DALAM FPGA

Secara traditional, algoritma pemprosesan isyarat digit dilaksanakan dengan menggunakan perisian kerana kerumitan untuk melakukan operasi yang terlibat. Dalam applikasi yang berspesifikasi tinggi, litar bersepadu applikasi spesifik (ASIC) digunakan. Kecepatan pemprosesan biasanya adalah berkadar langsung kepada kos perkakasan. Dengan FPGA baru yang murah dan berkeupayaan tinggi, pemprosesan digit secara perkakasan dapat dilaksanakan dengan murah. Sistem perkakasan yang berkeupayaan tinggi ini dapat digunakan dalam applikasi yang kritikal seperti pemprosesan imej perubatan untuk pengesanan parasit malaria. Imej dapat diproses secara masa nyata dan ini merupakan suatu peralatan yang penting untuk pempraktik perubatan. Projek ini akan mengaplikasi penapis median 3X3 untuk menapis hingar dari imej palitan darah dalam pengesanan parasit malaria. Sistem adalah dibuat dengan VHDL dan dibahagi kepada tiga blok. Setiap blok adalah dikawal dengan mesin keadaan dan operasi secara serentak. Dua daripada blok tersebut dibina dengan menggunakan arkitektur piawai yang diubah suai. Selepas simulasi perisian, perkakasan dibina dan keputusan menunjukkan sistem ini dapat melakukan penapisan median kepada imej perubatan. Sistem ini berupaya memproses sebarang imej dalam julat saiz tertentu dan mempunyai masa pemprosesan yang singkat.

ABSTRACT

Traditionally, digital signal processing algorithms are implemented using software because of the complexities involved in the operations. In high-demand applications, application-specific integrated circuits (ASICs) are used. Faster processing usually comes at higher cost. With new, low cost and powerful FPGAs, hardware based digital processing can become affordable. The powerful processing system can cater to critical application such as in medical imaging for malaria parasite detection. Images can be processed in real time and this would be a great tool for medical practitioners. This project implements a 3X3 median filter system for the removal of noise from blood smear images for malaria parasite detection. The system is described using VHDL into three building blocks. Each block is state machine controlled and operates simultaneously. Two of the blocks are built based on modified standard architectures. Upon completing the software simulations, the hardware is built and the results show that the system is able to perform median filter to medical images. This system is able to process any medical images up to a preset size and have short processing time.

ACKNOWLEDGEMENTS

This project would not have been possible without the support of many people. Many thanks to my superviser, Dr. Nor Ashidi Mat Isa, who read my numerous revisions and helped make some sense of the confusion. Also thanks to Pn. Zaini, lab technicians and friends who offered guidance and support. Thanks to the Universiti Sains Malaysia for providing me resources and means to complete this project. And finally, thanks to my family and friends who endured this long process with me, always offering support and help.

Thank you.

Ng Soo Kheng
Universiti Sains Malaysia
Engineering Campus
May 2006

CONTENTS **Page Number**

ABSTRACT ii

ACKNOWLEDGEMENTS iv

LIST OF CONTENTS v

LIST OF FIGURES viii

LIST OF TABLES x

ABBREVATIONS xi

CHAPTER 1 INTRODUCTION

1.1 Medical Imaging …….. 1

1.2 Project Objectives ……..... 2

1.3 Overview of Project ……. 3

1.3.1 Software Design …….. 3

1.3.2 Hardware Design ……. 4

1.4 Project Scope ……. 5

1.5 Chapter Overview …… 7

CHAPTER 2 LITERATURE REVIEW

2.1 Introduction …….. 8

2.2 Median Filter …… 8

2.2.1 Median Filter Technique ……. 8

2.2.2 MATLAB Implementation …….. 11

2.3 Finite State Machines (FSM) ……..... 12

2.3.1 Moore Machine …… 12

2.3.2 Mealy Machine ……. 13

2.4 NVRAM ……. 14

2.5 External RAM Protocol ……. 14

2.5.1 Read Protocol …… 14

2.5.2 Write Protocol ……... 16

2.6 Internal RAM Protocol …….. 17

2.7 Image Format in RAM …….. 18

2.8 Architecture …….. 19
2.8.1 Median Block …….. 19
2.8.1.1 Median Block Diagram …….. 20
2.8.1.1 Median Architecture …… 20
2.8.1.2 Median State Diagram …… 23
2.8.1.3 Median Machine Waveform …… 24
2.8.2 Fetch Block …….. 24
2.8.2.1 Fetch Block Diagram …….. 25
2.8.2.2 Original Fetch Architecture …… 25
2.8.2.3 Derived Fetch Architecture ……. 28
2.8.2.4 Advantages and Disadvantages …….. 30
2.8.2.5 Fetch Machine Flow Chart ……. 31
2.8.3 Write Block ……. 31
2.8.3.1 Write Block Diagram ……. 33
2.8.3.2 Write State Machine Flow Chart …… 33
2.9 Summary …… 34

CHAPTER 3 IMPLEMENTATION

3.1 Introduction …… 35
3.2 Software Design in VHDL ……..…… 35
3.2.1 Median Block …… 35
3.2.1.1 Median Components' I/O Ports …… 35
3.2.1.2 Median I/O Ports …….. 36
3.2.1.3 Implementation Solutions ……. 37
3.2.2 Fetch Block …… 39
3.2.2.1 Fetch I/O Ports …….. 40
3.2.2.2 Implementation Solutions ……. 41
3.2.3 Write Block …… 44
3.2.3.1 Write I/O Ports …….. 44
3.2.3.2 Implementation Solutions ……. 45
3.2.4 Overall Block Diagram ……. 46

3.3 Creating Hex File to Program RAM …… 46
3.4 Hardware Design …… 47
3.4.1 Hardware Specifications …… 47
3.4.2 Hardware Connections …….. 47
3.4.3 Protocol Test ……. 48
3.5 Programmer …… 49
3.6 Conclusion ……... 49

CHAPTER 4 RESULTS AND DISCUSSIONS

4.1 Introduction …… 51
4.2 Operation Flow Trace …… 51
4.3 Reports from Compilation …… 54
4.4 Simulation Results from MATLAB …….. 55
4.5 Result from QUARTUS II Simulation …….. 56
4.6 Hardware Result …… 57
4.7 Comparison between Simulation and Actual Results …... 58
4.8 Conclusion …… 58

CHAPTER 5 CONCLUSION

5.1 Introduction …….. 59
5.2 Suggestions …….. 59
5.3 Conclusion …… 60

REFERENCES 61
APPENDIX A: SYSTEM BLOCK DIAGRAM 63
APPENDIX B: PIN ASSIGNMENTS 64
APPENDIX C: SIMULATION RESULT 65
APPENDIX D: MEDIAN FILTERED IMAGES 69

LIST OF TALBES

Table	Title	Page number
2.1	RAM read protocol	16
2.2	RAM write protocol	17
3.1	Inputs and outputs list for median block components	36
3.2	I/O ports for the median machine	37
3.3	I/O ports for fetch machine	40
3.4	I/O ports for the write block	45
4.1	Operation flow	52

LIST OF FIGURES

Figure	Title	Page number
1.1	Block diagram showing the three main blocks of the image processing system	3
1.2	Hardware block diagram	4
1.3	Project design flow	5
2.1	An example of median filtered image	9
2.2	Two 3X3 mask on an image and their respective center positions	9
2.3	3X3 mask with covered pixels and the median value	10
2.4	Image after zeros' padding to boundaries	11
2.5	MATLAB image processing toolbox function	12
2.6	State diagram and state table for a Moore machine	13
2.7	RAM read timing waveform	15
2.8	RAM write timing waveform	16
2.9	RAM read waveform	17
2.10	RAM write waveform	18
2.11	Example of image to be saved in RAM	18
2.12	RAM content of image in Figure 2.11	19
2.13	Median block diagram	20
2.14	A median architecture by John L. Smith	21
2.15	Refined median architecture	22
2.16	Median state machine	23
2.17	State machine responses waveform for single input	24
2.18	State machine responses waveform for batch input	24
2.19	Fetch block diagram	25
2.20	FIFO for pixel arrangement	26
2.21	Example of 5X5 image for FIFO explanation	26
2.22	First position of mask during median filtering	27

2.23	Second position of mask during median filtering	27
2.24	FIFO ready for first median filtering	27
2.25	FIFO ready for second median filtering	28
2.26	Derived architecture for fetch block	29
2.27	5X5 image for derived architecture explanation	29
2.28	Contents of RAMs after filling the first three rows of data from Figure 2.27	29
2.29	RAMs' content after filling the forth row of image data	30
2.30	Fetch machine flow chart	32
2.31	Block diagram for the write machine	33
2.32	Flow chart of the state write machine	34
3.1	An example of coding style for the median pipeline	38
3.2	An example of component declaration and specifying connectivity	39
3.3	Vector_to_int function which converts *std_vector* type to *integer* type	42
3.4	An example of state machine code	44
3.5	Altera board connected to RAMs	48
3.6	Testing the RAM protocol	49
3.7	The programmer for saving original RAM image	50
4.1	A 5X5 image	51
4.2	Summary report from *Fitter*	54
4.3	Summary report from *Timing Analyzer*	54
4.4	A blood smear sample image	55
4.5	Image on the left is the original image and median filtered image is on the right	56
4.6	Histogram equalized of images from Figure 4.5	56
4.7	A 9X9 test image and the expected result after median filtering	57
4.8	Original image on the left and hardware processed image on the right	58

ABBREVIATIONS

FPGA	Field Programmable Gate Arrays
CPLD	Complex Programmable Logic Devices
VHDL	Very High Speed Integrated Circuit Hardware Description Language
CMOS	Complementary Metal-Oxide Semiconductor
RTL	Register Transfer Level
ASIC	Application Specific Integrated Circuit
NVRAM	Non-volatile Random Access Memory
EEPROM	Electrically Erasable Programmable Read Only Memory
SRAM	Static Random Access Memory
FSM	Finite State Machine

CHAPTER 1

INTRODUCTION

1.1 Medical Imaging

Medical imaging is the process by which physicians evaluate an area of the subject's body that is not normally visible. Medical imaging may be "clinical", seeking to diagnose and examine disease in specific human patients. (Wikipedia, 2006)

An image is defined as two-dimensional function, f(x, y), where x and y are spatial coordinates, and the amplitude of f at any pair of coordinate (x, y) is called the intensity or gray level of the image at that point. A digital image is where x, y and the amplitude values of f are all finite, discrete quantities (Gonzalez, 2002). Digital image processing can be divided into two categories: gray level processing and color image processing. This project only focuses on gray level processing. This means that the images to be processed are monochrome pictures.

More often than not, we need to process an image so that it will meet our requirements. Noise and poor contrast are usually the major problems of an image. These problems are even more critical in medical images as poor image would result in wrong analysis result and therefore wrong action be taken. Blood smear images of malaria parasites would be used to evaluate the performance of the FPGA based hardware image processor.

1.2 Project Objectives

This project aims to create a hardware based image processing system using FPGA to process blood smear images for malaria parasites detection. A 3X3 median filter is chosen for the implementation. VHDL will be used to describe the system. The targeted device is EPF10K70 in a 240-pin power quad flat pack (RQFP) package of the FLEX10K family from Altera.

There are three objectives to be achieved:

- Build a hardware based image processing system. The system should be able to process medical images of user defined size that does not exceed the hardware capabilities of the system. This means that this system must be capable of filtering images of any size within the range of the hardware limitation.

- The system would perform noise reduction, in specific impulse noise, using 3X3 median filter on medical images which are 8 bit gray level images. The filtered images must be suitable to be used for malaria parasites detection and help to distinguish between infected and healthy cells.

- To apply multistage or pipelined architecture to the system for improved performance. The system should have a decent performance with respect to its software counterpart.

1.3 Overview of Project

The project is divided into two design stages. They are software design and hardware design.

1.3.1 Software Design

The software part of the design has been completed and simulated to work in ideal situation. Quartus II software from Altera has been used in the development of the software design. The software provides all-in-one functions from compilation, synthesize, place and route, timing analysis, simulation to program download of hardware.

There are three major block for the design. The blocks are named as fetch, median and write which is shown in Figure 1.1.

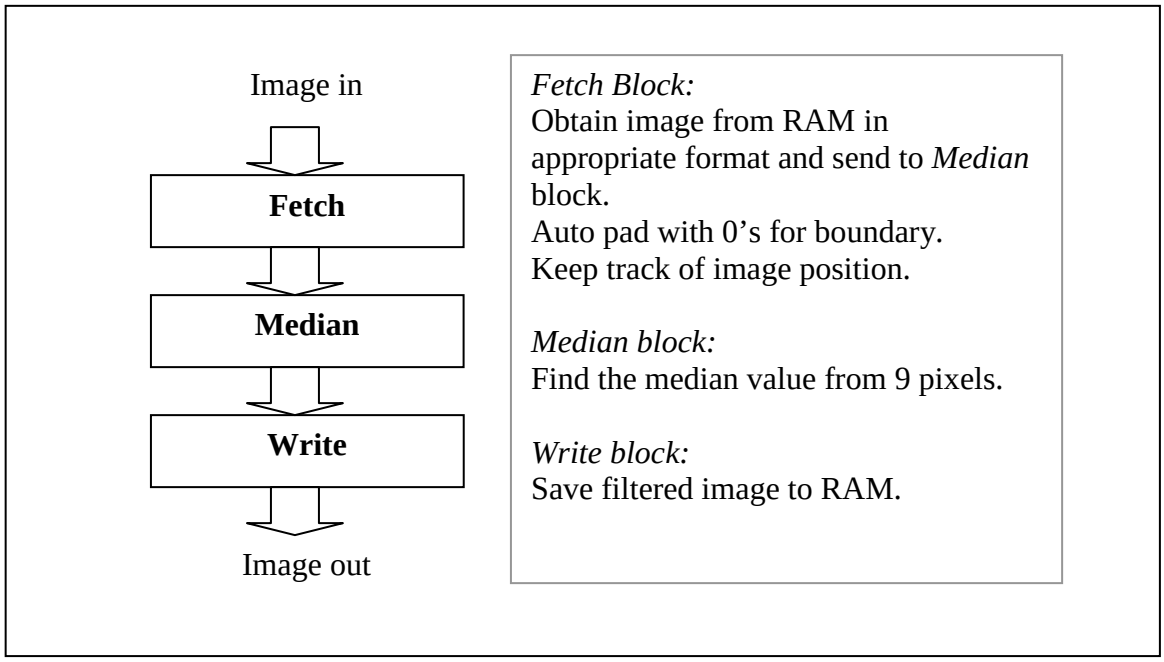


Figure 1.1: Block diagram showing the three main blocks of the image processing system.

The overall system would be a three stage pipelined median filter, with three blocks operating simultaneously. *Fetch* block would have to deal with memory interfaces, obtaining the nine pixels covered by the 3X3 mask, padding image boundary with 0's and a

program counter to keep track of the image position. *Median* block will only perform median filter of data's passed from fetch unit. Finally the *write* block will save the results.

1.3.2 Hardware Design

The University Program UP2 Education Kit is used to implement the system. The board consists of two devices, EPF10K70 and EPM7128S. Only EPF10K70 would be used. The EPF10K70 device is based on SRAM technology. It is available in a 240-pin RQFP package and has 3,744 logic elements (Les) and nine embedded array blocks (EABs). Each LE consists of a four-input LUT, a programmable flip flop, and dedicated signal paths for carry-and-cascade functions. Each EAB provides 2,048 bits of memory which can be used to create RAM, ROM, or first-in first-out (FIFO) functions (Altera, 2004). Hardware block diagram for the system is shown in Figure 1.2.

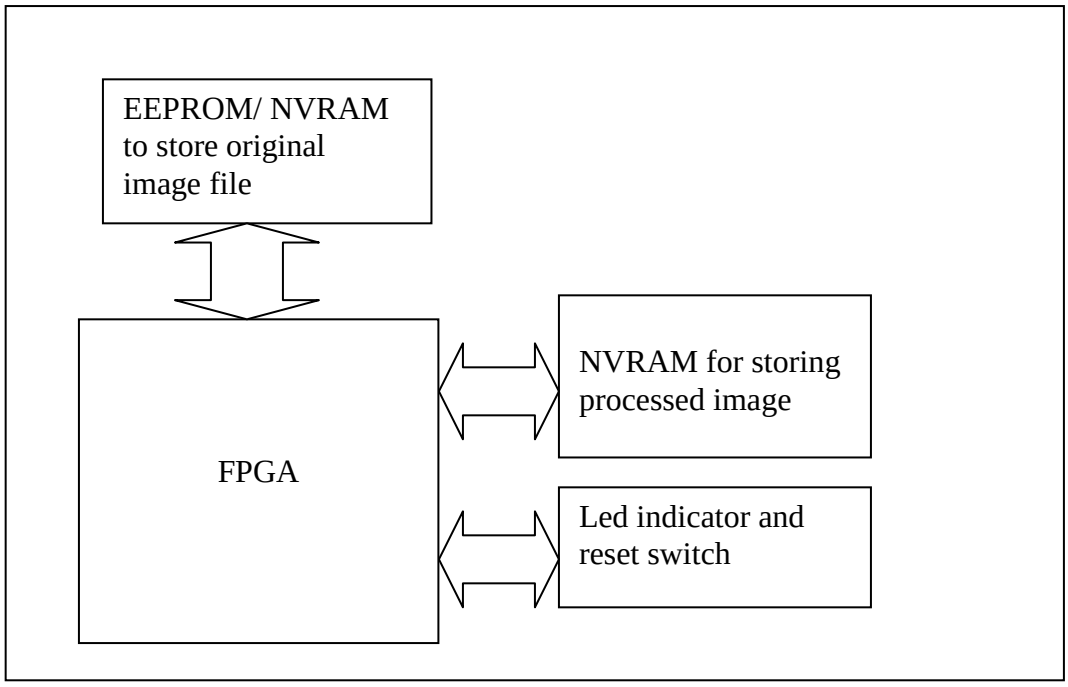


Figure 1.2: Hardware block diagram.

1.4 Project Scope

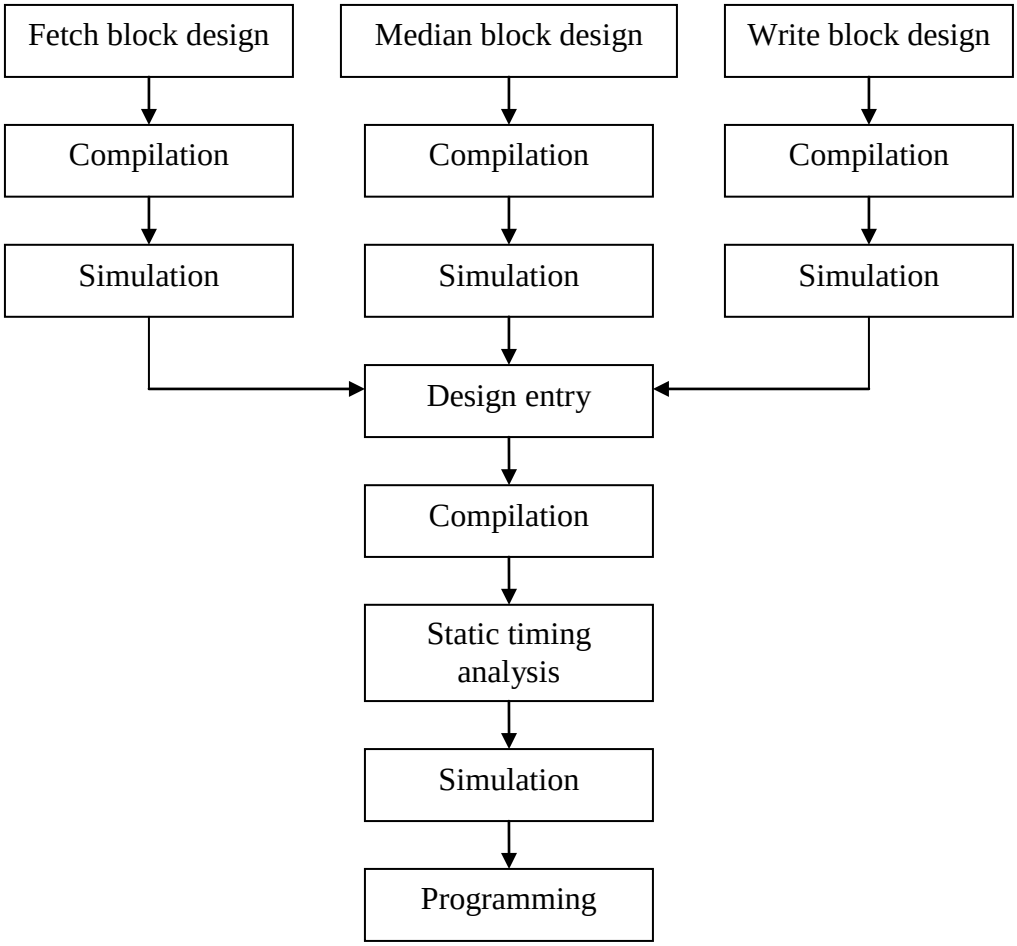


Figure 1.3: Project design flow.

The first step in designing the system is to describe the system in HDL language. This is the design entry phase. The three blocks; namely fetch, median and write blocks, are first described independently. They are then compiled and simulated to ensure proper operation. Upon simulation, a new project is created and a block diagram is drawn for the whole system. The three blocks are combined in this project and interconnection signals are added to allow synchronization of the three blocks. The inputs and outputs are also

assigned to the desired locations in Assignment Editor. The project is then compiled using Quartus II compilation module.

The Quartus II Compiler consists of a set of independent modules that check the design for errors, synthesize the logic, fit the design into an Altera device, and generate output files for simulation, timing analysis, software building, and device programming. During a full compilation, the Compiler's Analysis & Synthesis module first extracts information that defines the hierarchical connections between a project's design files and checks the designs for basic design entry errors. Then, it creates an organizational map of the design and combines all design files into a flattened database that can be processed efficiently. Next, the Fitter selects the optimum interconnection paths, pin assignments, and logic cell assignments needed to fit the design into the selected Altera device. The Assembler then completes project processing by converting the Fitter's assignments into a programming image for the device. Finally, the Timing Analyzer runs automatically to report timing information for all logic in the design.

In the static timing analysis phase, we view the report from Timing Analyzer. The report specifies the specifications of the system which include the maximum clock frequency, clock setup and hold time. The report must not have error and must meet the minimum requirement for the system to be operational. For example, the maximum clock frequency, Fmax. Fmax in the report must be higher than 25.175 MHz because University Program UP2 Education Kit is being used. The board has an on board oscillator that drives a global clock input which is clocked at 25.175 MHz. If somehow a lower clock speed is needed, an external clock input would be needed.

Once all requirements are met, the system is simulated to ensure proper operation. Finally, the programming file can be downloaded into the device using ByteBlaster II when all hardware connections are done.

1.5 Chapter Overview

Chapter 1 gives an introductory overview to the project which includes the objectives of the project, software and hardware introduction and the project's scope. Chapter 2 gives a literature review on information and researches related to the project. The architectures used in the project are also discussed in this chapter. Chapter 3 gives an overview on how the system is implemented in the Quartus II software. Solutions to implement the system in software design stage are also given. Chapter 4 shows the results of the project and discuss on them. This chapter compares the results obtained from actual system to results from simulations. Chapter 5 concludes the project.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

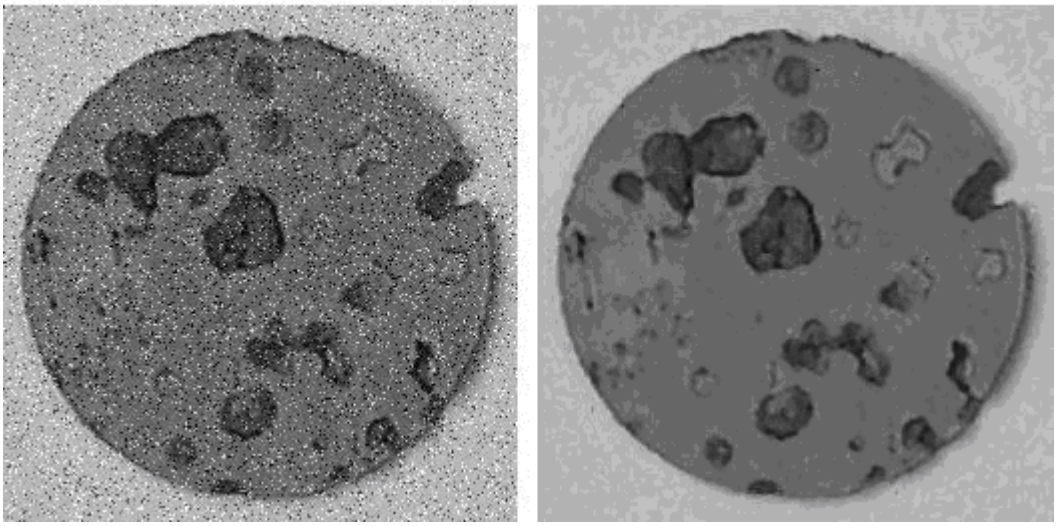
This project aims to create a hardware based system that can perform 3X3 median filter on medical images for malaria parasites detection. The medical images would be 8 bit gray level pictures. In 8 bit image, each pixel is represented by 256 levels of brightness. The greater the value of the pixel is, the brighter the pixel. The value 255 represents white and the value 0 represents black. This chapter will explain the details of studies and researches that are useful in understanding and implementing the system.

2.2 Median Filter

Median filter is the best-known order-statistics filter. Order-statistics filters are nonlinear spatial filters. They operate by ordering the pixels contained in the image area encompassed by a mask and then replacing the value of the center pixel with the value determined by the ranking result. In the case of median filter, the median pixel enclosed by a mask is replaced by the median of the gray levels in the mask.

2.2.1 Median Filter Technique

Median filter provides excellent noise-reduction capabilities with considerable less blurring than linear filters of similar size. They are particularly effective in filtering impulse noise, also called salt-and-paper noise. This noise appears as black and white dots superimposed on an image. Figure 2.1 shows an example on how a median filter can effectively removes impulse noise.



(a) (b)

Figure 2.1: An example of median filtered image. (a) Original image (b) Resultant image (http://tracer.lcc.uma.es)

In order to perform median filtering, we need to sort the values of the pixel contained in a mask and assign this value to the center pixel of the mask. The median value, ξ, would then have a value such that half the values in the set are less than or equal to ξ, and half are greater than or equal to ξ. Figure 2.2 shows two 3X3 masks and the targeted center pixel where the median will replace. The mask can be of any size, be it 3X3 or 5X5.

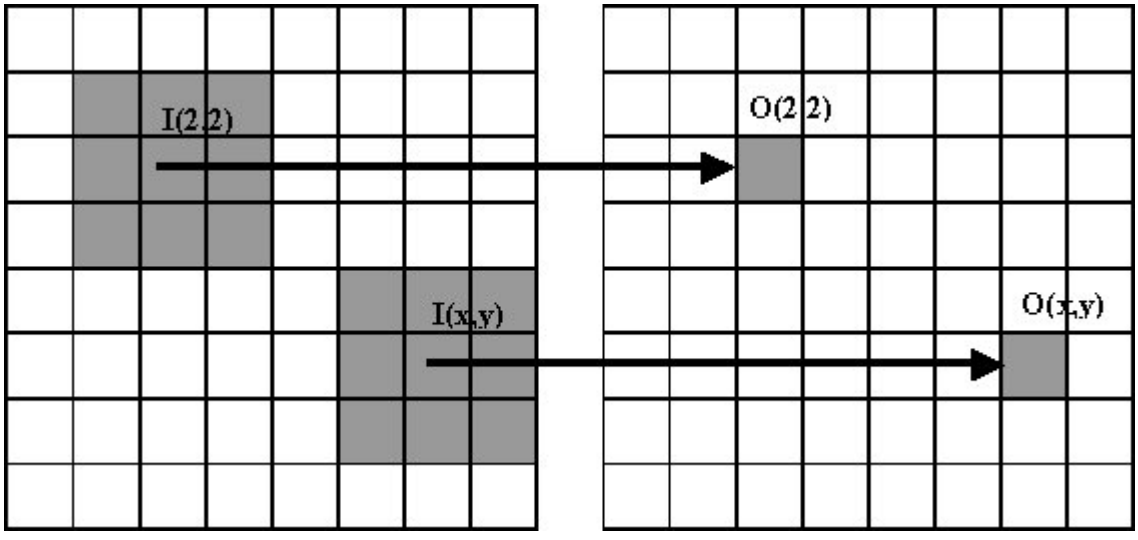


Figure 2.2: Two 3X3 mask on an image and their respective center positions. (http://tracer.lcc.uma.es)

Imagine, we have a 3X3 mask covering a portion of an image and the values covered are 10, 30, 5, 20, 200, 20, 15, 10 and 30. The mask and the values are shown graphically in Figure 2.3. Using these values, we arrange the values in ascending or descending order and finally get the median value that is positioned in the middle. This is also shown in Figure 2.3.

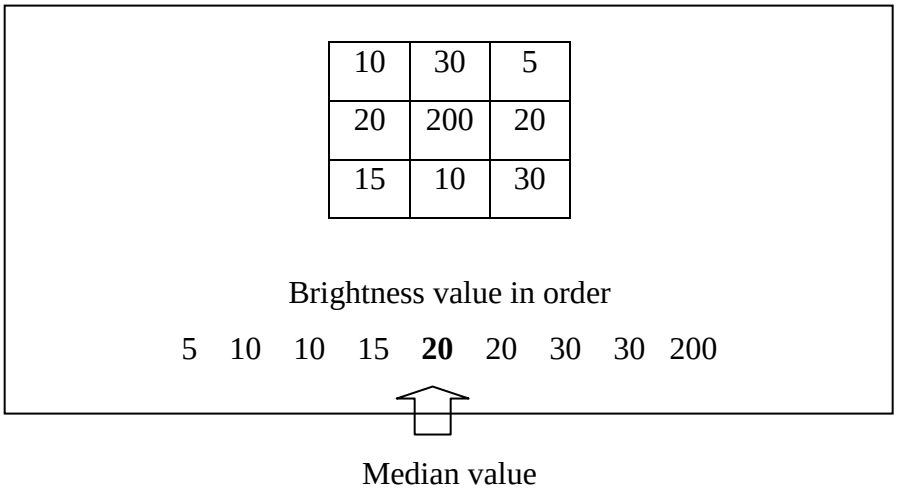
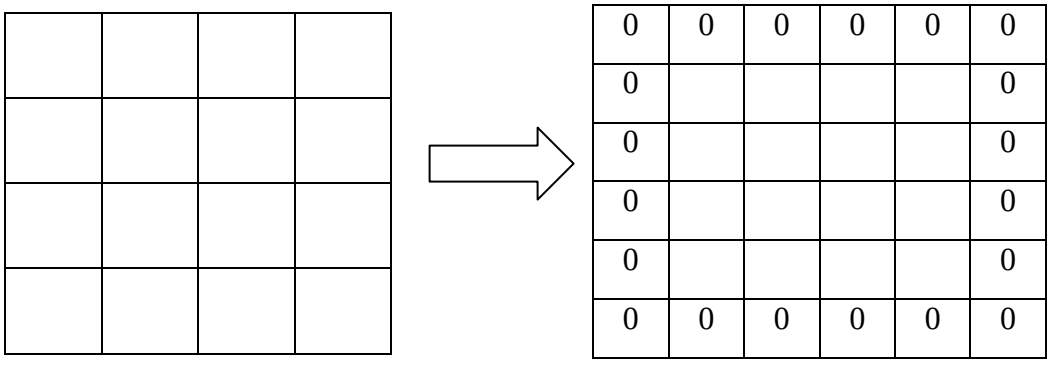


Figure 2.3: 3X3 mask with covered pixels and the median value.

When we apply a mask, for example 3X3 mask, on an image, the pixel at the boundary does not have the full nine neighborhood pixels. The usual way to solve this is to pad the boundaries of the image with zeros and this solution would be used in the system. Imagine a 4X4 image that we would like to pad with zeros. After padding the zeros to the boundary the image, we would get a 6X6 image. The result is graphically shown in Figure 2.4.

Note that this solution is only good for 3X3 median filter. If 4X4 median filter is being used, then we need to pad the boundaries so that the final image would become 8X8 in size from the original size of 4X4 image.



Original image Padded image

Figure 2.4: Image after zeros' padding to boundaries.

2.2.2 MATLAB Implementation

Before jumping straight into designing a median filter, we need to test out the theories behind the technique. This step is important because it allows us to confirm if this method is suitable for the medical images in malaria parasites detection; which is one of the objectives of the entire project. Also important, is the need to confirm the algorithm of the median filter. We do not want to begin implementing the system half way only to find out that we are using the wrong algorithm or the method to be implemented is not the correct one. With software simulation of the median filter, we would have the results which can be referred to when implementing the system.

A function is created in MATLAB software called *my_median*. First, the function will save the image to a variable called *input_image.* The data must be raw eight bit gray level image. The size of the image would be checked by determining the row and column of the data. An empty variable called *temp_image* is created to the size of padded image and then the original image is copied to the *temp_image*. This will automatically pad the boundaries with zeros'. Using *median* function in Matlab, we find the median for the 3X3

mask that we applied to the whole image and save the results to *output_image*. Finally, we send the result as output after removing the extra boundaries.

```
I = imread('eight.tif');
K = medfilt2(J);
imview(J), imview(K)
```

Figure 2.5: MATLAB image processing toolbox function.

Apart from that, there is also an image processing toolbox in MATLAB. The toolbox supplies a median filter function. Comparing the results of both functions proved that the algorithm to be used in the system is in fact correct.

2.3 Finite State Machines (FSM)

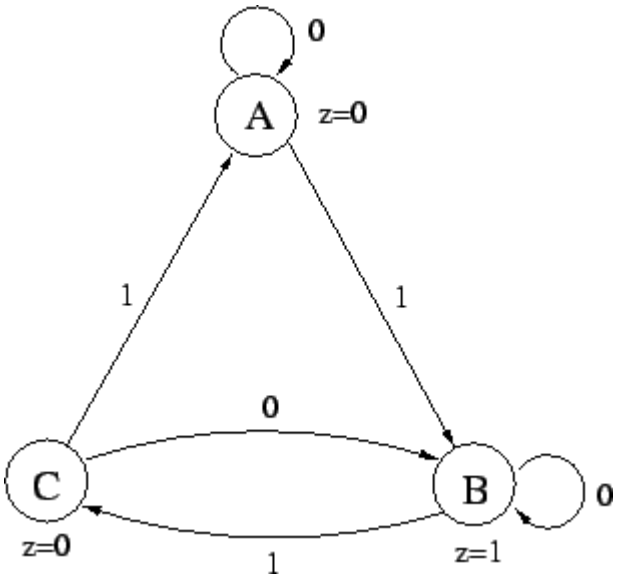
State machines are usually the simplest way to design a system. The method provides clear picture of the system in term of states and has documented design methodology.

2.3.1 Moore Machine

A state machine which uses only *Entry Actions*, so that its output depends on the state, is called a Moore model. (Wagner, 2005).

From Figure 2.6, it can be seen that all the transitions into state A and C are assigned the output 0, while all the transitions into state B are assigned the output 1. The output value 0 can be associated state A and state C while output value can be associated to state B. This type of machine is called Moore machine because each state has a certain pre-assigned output value. Because of this, the output value of the system only depends on the

states. The input on the order hand, will only effect the change in state. Defining the initial state in a Moore machine determines directly the first symbol of the output sequence. (Federico, 2004).



PS	NS $x=0$	NS $x=1$	$Output$
A	A	B	0
B	B	C	1
C	B	A	0

Figure 2.6: State diagram and state table for a Moore machine. (Federico, 2004)

2.3.2 Mealy Machine

A state machine which uses only *Input Actions*, so that the output depends on the state and also on inputs, is called a Mealy model (Wagner, 2005). When a beginning state is chosen in a Mealy machine, the output sequence is undefined until the first transition occurs. An example of a Mealy machine is a string recognizer. This project will also utilize the Mealy machine because it needs to read the header file of an image.

2.4 NVRAM

NVRAM is the abbreviation of Non-Volatile Random Access Memory. It is a type of memory that retains its contents when power is turned off. This definition is given by Webopedia.

The type of NVRAM used in this project is SRAM that is made non-volatile by connecting it to a constant power source; in this case, a battery. The NVRAM used in this project is DS1230Y-70 manufactured by Dallas Semiconductor. It is a low power CMOS type and is 256k bit in size. It also has a low 70ns read or write cycle time.

2.5 External RAM Protocol

This system will utilize two external memories. One memory will be used to store the original image data and the other is for storing the result. The memory for storing the result must be NVRAM because it has simple protocol. Flash EEPROM would be used for storing the original image data because the lack of additional NVRAM at the time of project progress. Read protocol applies to both NVRAM and Flash EEPROM. Write protocol only applies to the NVRAM.

2.5.1 Read Protocol

Before beginning on the fetch block, it is necessary to determine the RAM protocol so that the device can be properly interfaced to the RAM. Wrong timing would read the wrong data. If there signal contention at the data pins, either the RAM or the FPGA could be damaged. Based on the RAM's datasheet, the protocol for reading and writing was determined. The fetch and write block will follow the fixed protocol when reading and writing to RAM. This is highly critical because without the correct timing, the system could not get the correct data and therefore no result can be obtained. Note, in deriving the read and write protocols, the system is assumed to be running at 50 MHz clock speed.

Figure 2.7 shows the extracted timing waveform from the NVRAM datasheet. Let's take t_{rc} as an example. Beside the t_{rc} notation, *70ns* is the specified minimum time required for the read cycle time. The notation *USE 120ns* is the specified protocol time to be implemented in system.

All requirements are being considered including the input pin delay time in the protocol. For that reason, the read time for the system is 120ns at the speed of 50 MHz clock input is relatively slow with respect to the fast 70ns read cycle of the NVRAM. This protocol is only applicable to system running at 50 MHz and below.

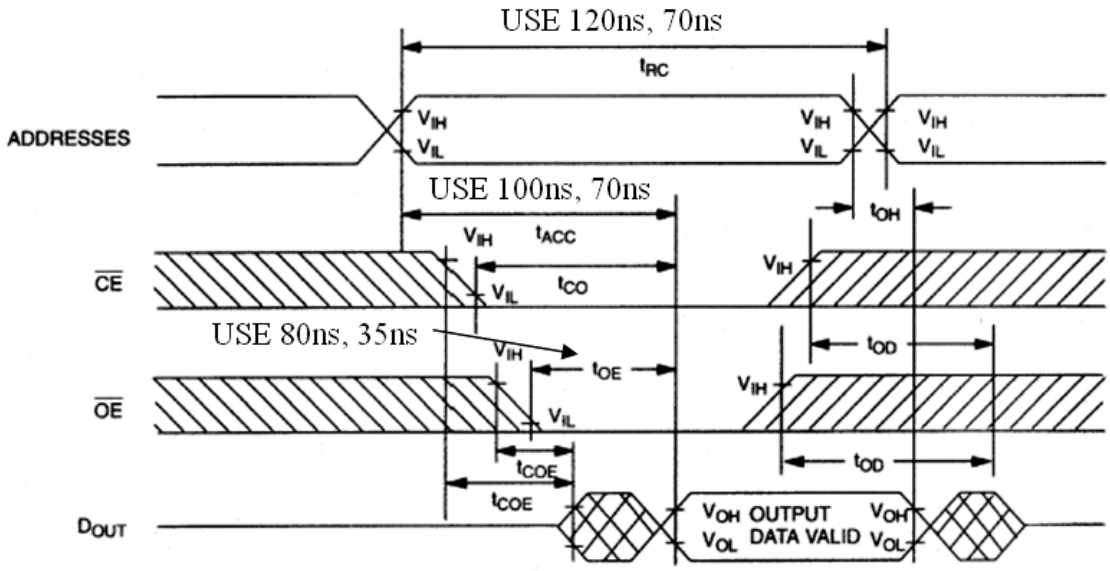


Figure 2.7: RAM read timing waveform.

Table 2.1 describes the RAM read protocol. The number from 1 to 7 represent the clock cycle from 1 to 7. Beside is the action to be done in the cycle.

Table 2.1: RAM read protocol.

1	Latch out address
2	Enable output
3	N/A
4	N/A
5	N/A
6	Latch data out
7	Disable output

2.5.2 Write Protocol

There is also a write protocol for the write block. Figure 2.8 shows the write timing waveform.

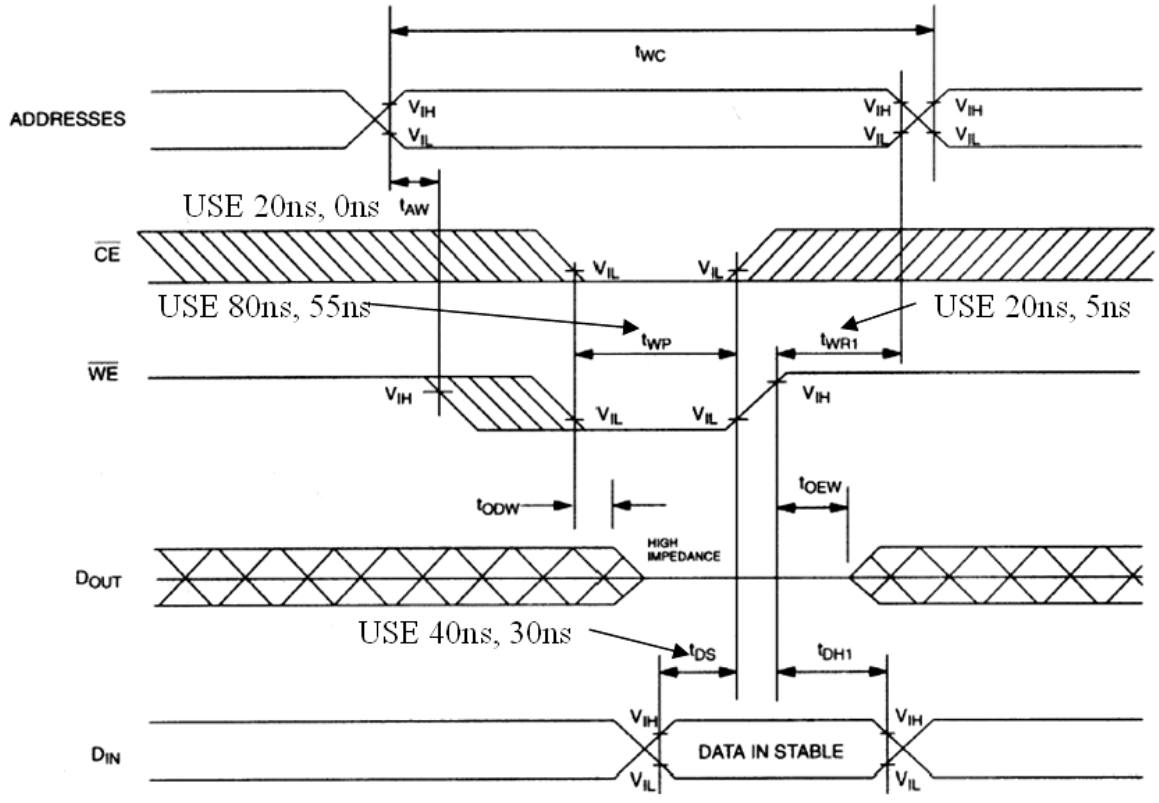


Figure 2.8: RAM write timing waveform.

Similar to the read protocol section, the protocol for writing to RAM is shown in Table 2.2.

Table 2.2: RAM write protocol.

1	Latch out address
2	Enable write
3	N/A
4	FPGA drive out data to be written
5	N/A
6	Disable write
7	FPGA data pin in high impedance

2.6 Internal RAM Protocol

Protocols for internal RAM also have to be determined so that they can be used properly. The internal memory modules are obtained from Altera library files. Batch read and write would be used in the system because of the nature of the architecture. Figure 2.9 shows the RAM read waveform and Figure 2.10 shows the RAM write waveform.

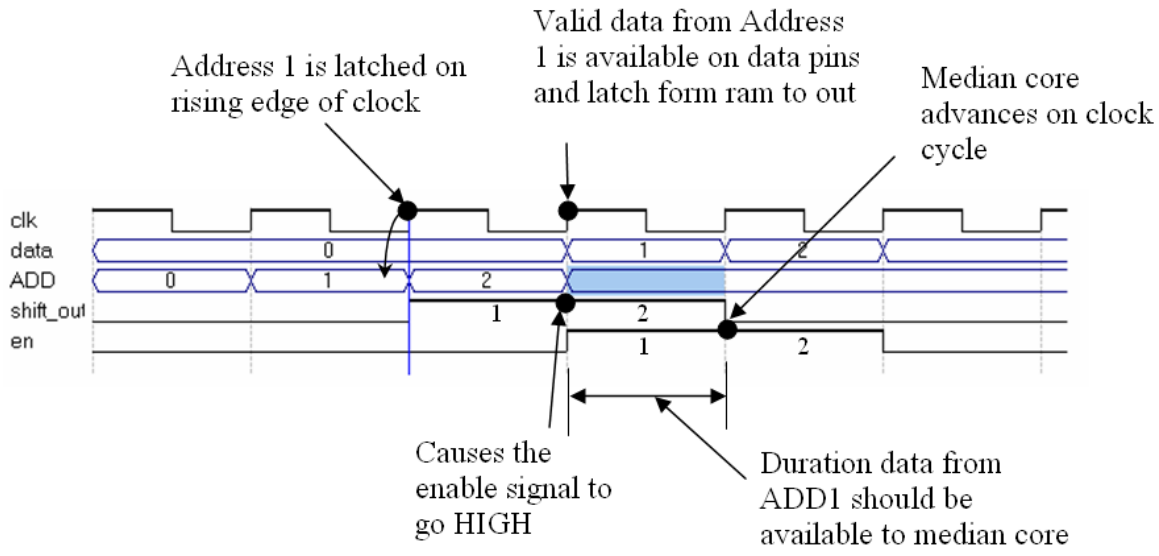


Figure 2.9: RAM read waveform.

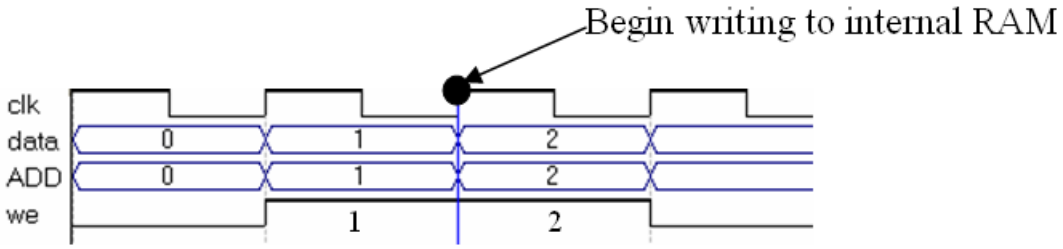


Figure 2.10: RAM write waveform.

2.7 Image Format in RAM

There is need to have a convention to store an image to a RAM. We would be storing the image data in raw format, in other words, there is no compression done on the image. In order to illustrate the format, let us consider Figure 2.11. Imagine we have a 3X4 image with values as shown in Figure 2.11, the image data would then be as shown in Figure 2.12.

51	74	4	152
7	111	255	55
97	79	23	3

Figure 2.11: Example of image to be saved in RAM.

The first two bytes will store the image height and the next two bytes will store the image width. These four bytes represent the header for the example image. The system would then have to determine the position for each pixel based on the header file.

ADD	CONTENT
0	3
1	0
2	4
3	0
4	51
5	74
6	4
7	152
8	7
9	111
10	255
11	55
12	97
13	79
14	23
15	3

Figure 2.12: RAM content of image in Figure 2.11.

2.8 Architecture

This section describes the architectures for the three blocks used in the project. The three blocks are median block, fetch block and write block. Median block is first described followed by fetch block and lastly the write block.

2.8.1 Median Block

The median block is the core function operator for the system. This block will find the median value of the three successive sets of three values passed it from the fetch block.

2.8.1.1 Median Block Diagram

Median block diagram is shown in Figure 2.13. Data in are the sets of three data inputs to the pipelined median core processor. Data out is the median value of data in. Control signals control the operation flow of the machine.

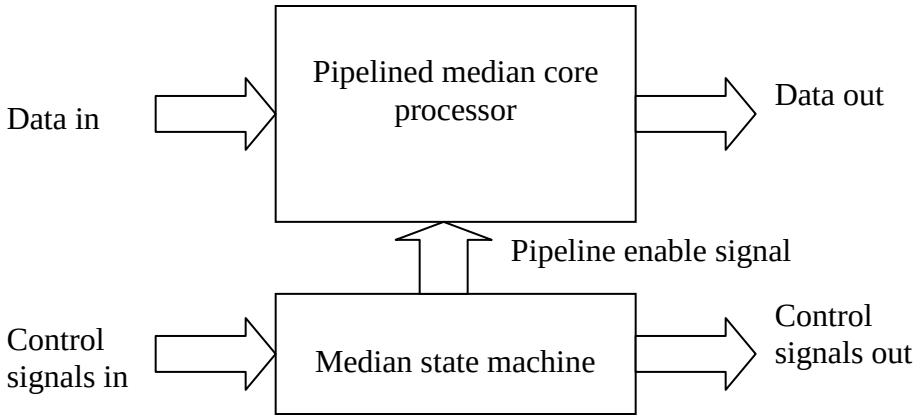


Figure 2.13: Median block diagram.

2.8.1.1 Median Architecture

The median architecture by J. L. Smith from Univision Techonologies Inc. would be used in the system because it offers a pipelined processing.(Xilinx). With the pipelined process, 3 data from 3X3 mask on an image can be fed to the median block continuously. The structure is implemented as 10 stages pipelined median core. This structure will output the median of 9 pixels, shown as P1 to P9 in Figure 2.14, with an output latency of 9 clock cycles. This means that only after 10 clock cycles of 3 data input sets, a valid result will be available at the output. The stages are named as arrange1 to arrange10.

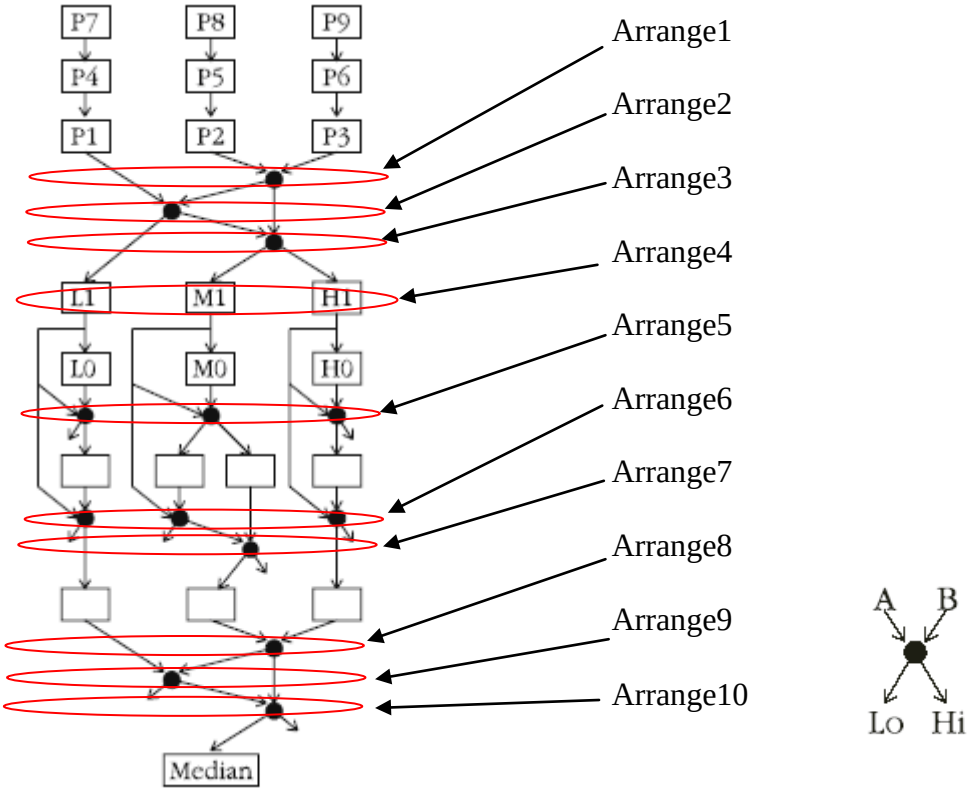


Figure 2.14: A median architecture by John L. Smith. (Xilinx)

It should be noted that A and B data are compared and the higher value of the two will always go to the right and the lower value to the left. Arrange 1 to arrange 3 would sort the three data so that the lowest value is on the left and highest value on the right for arrange 4 and arrange 5. Arrange 6 to arrange 10 would then use the pre-sorted data to find the median of the 3 sets sequence of 3 data.

Figure 2.15 shows a more refined flow of the pipelined structure that is more useful VHDL description. The block named out1, out2 to median are the names used for each stages' outputs. There are only 9 unique stages, because arrange1 and arrange3 are the same. These are then declared in a VHDL as components and connected using signal interconnections. These signals are presented as s1a, s1b to s9b as in the figure. For example, S1a connects out1 of arrange1 to the in1 of arrange2.

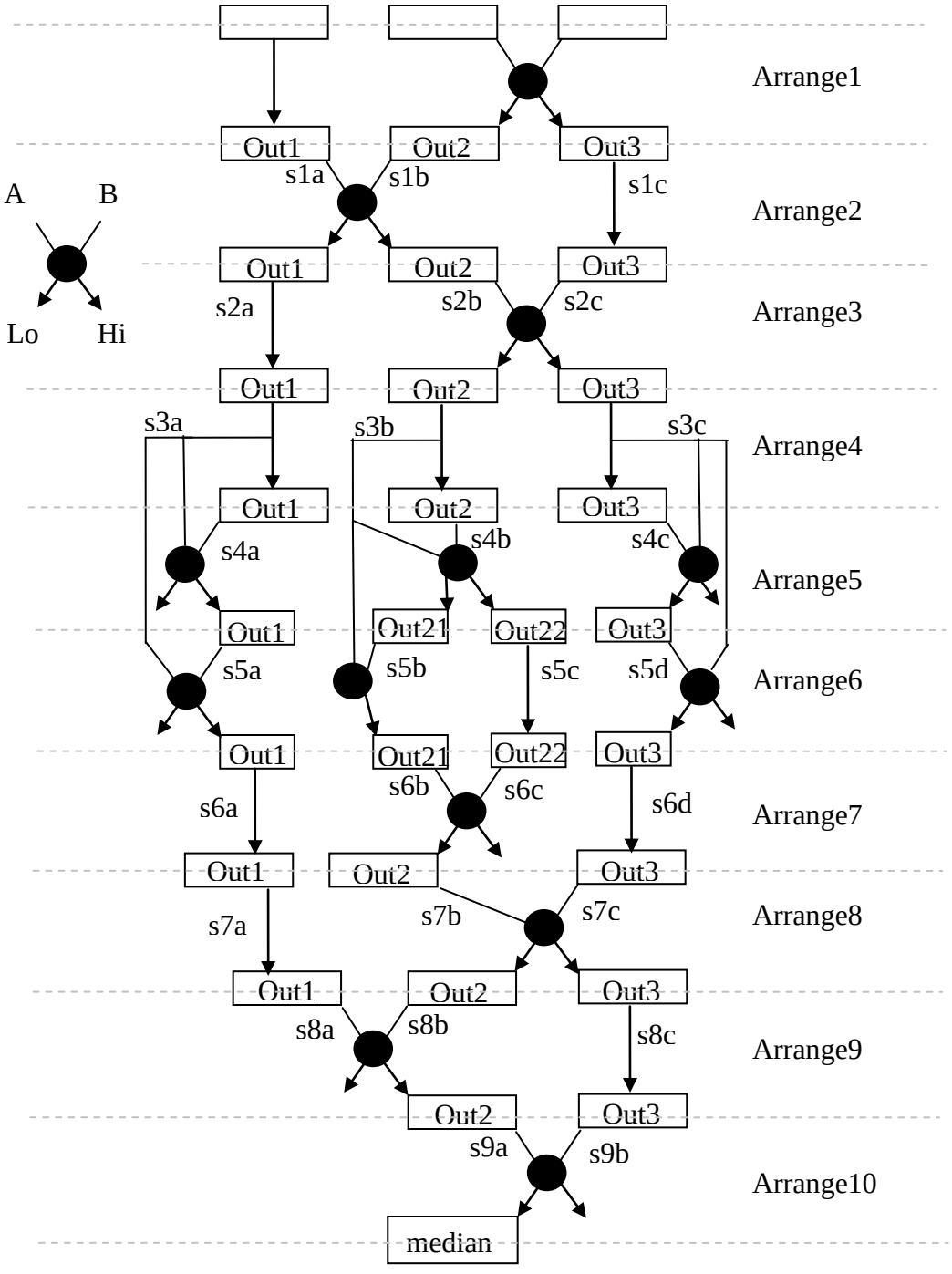


Figure 2.15: Refined median architecture.

2.8.1.2 Median State Diagram

It is decided that there would be a need to use state machine to control the pipeline and the inter-block signals. Figure 2.16 shows the state machine for the median block. Observing the figure, we know that this is a Moore machine because the outputs are associated with the states.

The machine consists of three states. They are *idle*, *clock_enable* and *initiate*. State *idle* means that the pipeline is not processing any data even at the rising edge of clock input. Both *clock_enable* and *initiate* enable the pipeline to clock input. *Initiate* is to differentiate when the first data from a row is fed to the input.

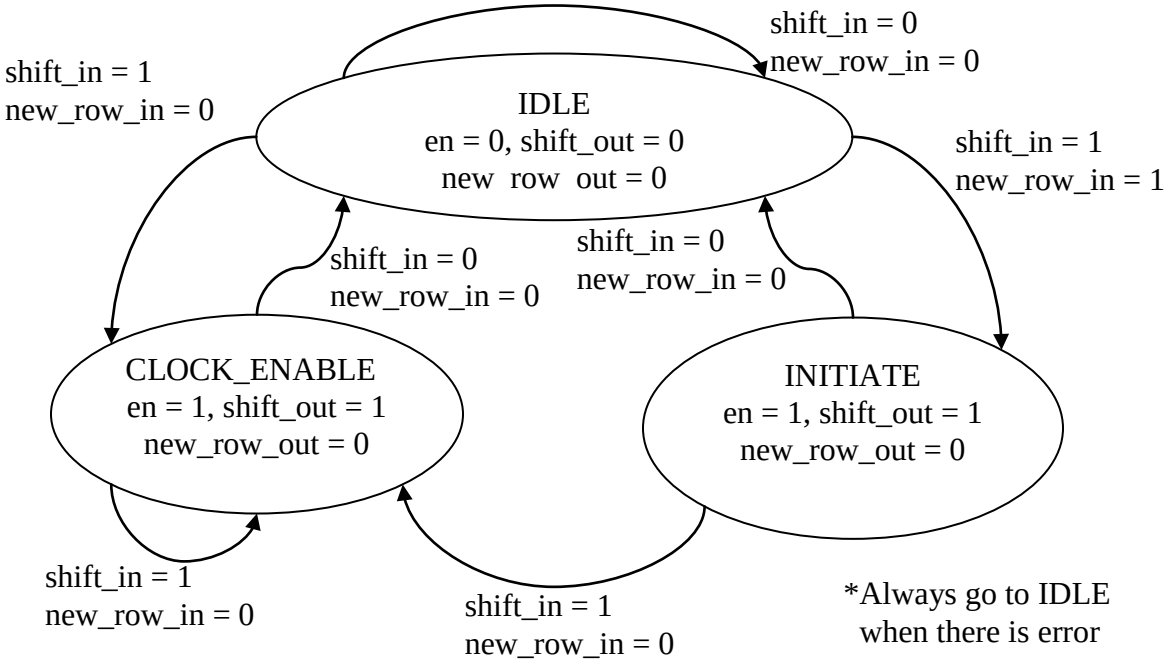


Figure 2.16: Median state machine.

2.8.1.3 Median Machine Waveform

Two expected waveform for the median machine were drawn for reference purposes during implementation. These waveforms are used to verify the communication signals for the state machine.

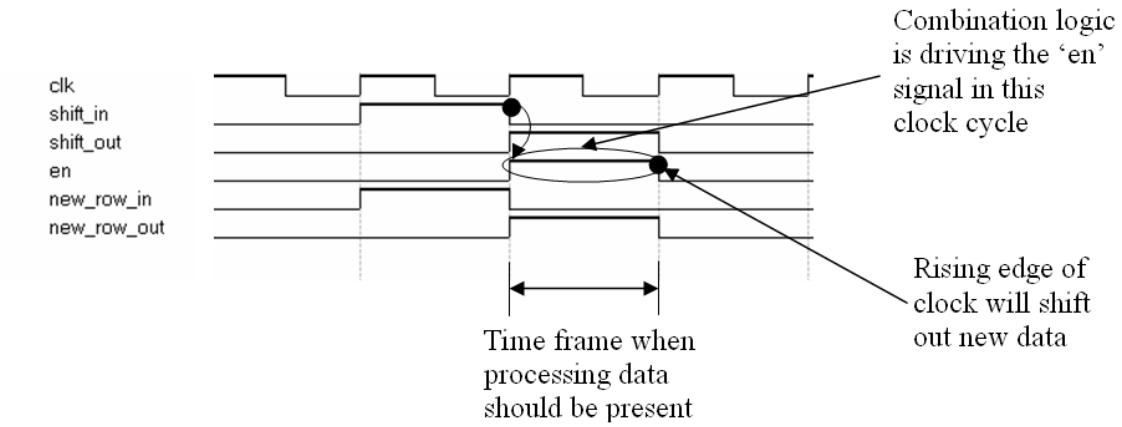


Figure 2.17: State machine responses waveform for single input.

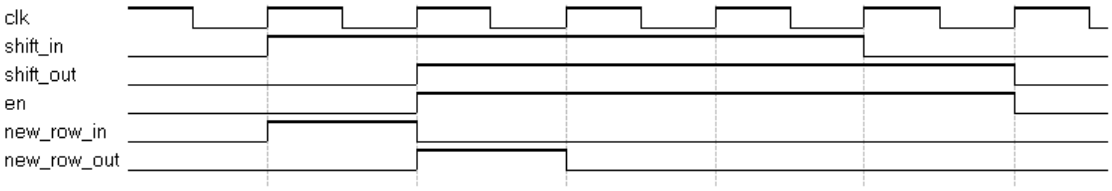


Figure 2.18: State machine responses waveform for batch input.

2.8.2 Fetch Block

The fetch block will arrange the image data into a proper format so that it is able to perform median filter as in applying a mask on the image in the software solution.