Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Yan Sun

Entitled
3D Image Segmentation Implementation on FPGA using EM/MPM Algorithm

For the degree of Master of Science in Electrical and Computer Engineering

Is approved by the final examining committee:

Lauren Christopher
Chair

Maher E. Rizkalla

Paul Salama

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Lauren Christopher

Approved by: Yaobin Chen 12/07/2010
Head of the Graduate Program Date

Graduate School Form 20
(Revised 9/10)

PURDUE UNIVERSITY
GRADUATE SCHOOL

Research Integrity and Copyright Disclaimer

Title of Thesis/Dissertation:

3D Image Segmentation Implementation on FPGA using EM/MPM Algorithm

For the degree of Master of Science in Electrical and Computer Engineering

I certify that in the preparation of this thesis, I have observed the provisions of *Purdue University Executive Memorandum No. C-22,* September 6, 1991, *Policy on Integrity in Research.**

Further, I certify that this work is free of plagiarism and all materials appearing in this thesis/dissertation have been properly quoted and attributed.

I certify that all copyrighted material incorporated into this thesis/dissertation is in compliance with the United States' copyright law and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and save harmless Purdue University from any and all claims that may be asserted or that may arise from any copyright violation.

Yan Sun

Printed Name and Signature of Candidate

12/07/2010

Date (month/day/year)

*Located at http://www.purdue.edu/policies/pages/teach_res_outreach/c_22.html

3D IMAGE SEGMENTATION IMPLEMENTATION ON FPGA USING EM/MPM ALGORITHM

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Yan Sun

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

December 2010

Purdue University

Indianapolis, Indiana

To my family

ACKNOWLEDGMENTS

Foremost, I would like to express my sincere gratitude to my advisor Prof. Lauren Christopher of the Department of Electrical and Computer Engineering, for the continuous support of my study and research, for her patience, motivation, enthusiasm, and immense knowledge. Her guidance helped me in all the time of research and writing of this thesis.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Paul Salama and Prof. Maher Rizkalla for their encouragement, insightful comments, and hard questions.

My sincere thanks also goes to Prof. Brain King, for offering the great help on my thesis writing.

My love also goes to Yuhui Sheng, Yu Ding, Chenyuan Feng, and Jinming Shao, my best friends. Without their support, I even can not got the opportunity to come to USA and continue my study. They are always besides me. And I know they will be there forever.

Last but not the least, I would like to thank my family: my parents Xiaobin Sun and Weiwei Li, for giving birth to me at the first place and supporting me spiritually throughout my life.

TABLE OF CONTENTS

Page

LIST OF FIGURES

ABSTRACT

Sun, Yan. M.S.E.C.E., Purdue University, December 2010. 3D Image Segmentation Implementation on FPGA using EM/MPM Algorithm. Major Professor: Lauren Christopher.

In this thesis, 3D image segmentation is targeted to a Xilinx Field Programmable Gate Array (FPGA), and verified with extensive simulation. Segmentation is performed using the Bayesian algorithm of Expectation-Maximization with Maximization of the Posterior Marginals (EM/MPM). This algorithm segments the 3D image using neighboring pixels based on a Markov Random Field (MRF) model. This iterative algorithm is designed, synthesized and simulated for the Xilinx FPGA, and greater than 100 times speed improvement over standard desktop computer hardware is achieved. Three new techniques were the key to achieving this speed: Pipelined computational cores, sixteen parallel data paths and a novel memory interface for maximizing the external memory bandwidth. Seven MPM segmentation iterations are matched to the external memory bandwidth required of a single source file read, and a single segmented file write, plus a small amount of latency.

1. INTRODUCTION

Due to its significant advantages in visualization, 3D images are becoming more and more popular in several aspects of our lives. On the one hand, in the medical area, because of the complexity and diversity of human organs as well as the unpredictable location of lesions, it is difficult to obtain accurate and complete tissue segmentation from 2D images. On the other hand, 3D images offer us three perpendicular planes simultaneously which can be rotated and translated in order to get accurate information and the suitable view the doctors need. For tissues surrounded by layers of different texture in some hidden angle, segmented 3D images in the visualization can improve clinical understanding. Therefore, segmented 3D images can help doctors view 3D rendered tissues and organs for diagnosis, treatment planning, and even surgical assistance in the operating room.

Several 3D image segmentation algorithms have been published recently. Among them, the Expectation-Maximization with Maximization of the Posterior Marginals (EM/MPM) algorithm is a good segmentation strategy, especially in noisy data [1] [2] [3]. The EM/MPM algorithm is a combination of EM algorithm for parameter estimation and MPM algorithm for segmentation. The MPM algorithm at first classifies every pixel and assigns a cost to the number of misclassified pixels, and then minimizes the cost to get segmentation of image. The EM algorithm iteratively estimates the model parameters to get the best probabilistic solution which is closest to the true value of model parameters. High resolution pixel volumes in 3D images results in Gigabytes of data to process. So the standard computing architectures are not well suited to the task due to fixed memory bandwidth and large instruction set overhead.

Because of the large data volume of 3D images and the iterative processes of pixel-based segmentation algorithm, on-chip system implementation for this 3D im-

age segmentation algorithm is proposed. Hardware implementations on FPGA and Application Specific Integrated Circuits (ASICs) have distinct advantages especially for a specific task with large data sets. On-chip systems can have significant parallelism to optimize repeated data processing. Some 3D medical imaging tasks have been mapped to hardware in the research literature. Li [4] presented a brick caching scheme for 3D medical imaging aiming at speeding up the processing on an FPGA. His work implied that parallel memory access and brick pre-fetching can be possible, but some ideas were left for future study. Others use a PCI-board with 8 RISC processors to do 3D image analysis. A parallel processor array for filtered back projection was developed in [5] to speed up processing. I.Goddard et al. [6]did high-speed cone-beam reconstruction based on embedded systems approach and S.Coric et al. [7] did parallel-beam back projection which is implemented in an FPGA platform for medical imaging. Accelerated volume rendering and tomographic reconstruction are demonstrated by B.Cabral [8] using texture mapping hardware. K.Mueller et al. [9] did fast and accurate three-dimensional reconstruction from cone-beam projection data using algebraic methods in his PhD dissertation. P.V.Dillinger [10] et al. propose a parallelizable 3D grey-value structure code for image segmentation on FPGA which can process segmentation in real time. K.J.Shanthi et al. [11] used histogram for image segmentation and implement this algorithm on FPGA which renders the algorithm more useful for real time application. S.B. Malarkhodi et al. [12] did the image segmentation work using Expectation-Maximization algorithm based on Gabor filter. Then they developed and coded the whole architecture using VHDL (very high speed hardware description language) to implement the design on SPARTAN-3E FPGA. M.A.Salem et al. [13] proposed a hardware implementation of the 2D wavelet transform which can reduce the computing power and memory requirements for video segmentation and movement detection. However, hardware implementation has its own limitations. First of all, although on-chip system can implement several processing cores to accelerate large volume data calculation, the speed of the I/O interface for the large volume data transmission is the greatest speed limitation for whole system.

Secondly, there are limitations for on-chip resources on different sizes of FPGA. For example, some FPGAs contain numerous DSPs but less on-chip memory for users. Some contain more memory resources but fewer look up tables (LUTs) on chip. So the balance of the different on-chip resources and the best arrangement of internal and external memory to minimize resource cost are the design challenges.

The work described in this thesis is important for the following reasons. First, this research is the first hardware FPGA implementation of the EM/MPM algorithm. Second, the method of parallel processing the volume data is unique. By generating multiple computational cores on chip, the on-chip data pipelining and parallelism handles the overlapping pixel neighborhoods automatically. Third, the new method of optimizing the iterative algorithm between on-chip and off-chip memory lowers the overall memory bandwidth and increases the processing speed by minimizing external memory accesses.

In chapter 2, the EM/MPM algorithm is reviewed. A global view and analysis of the algorithm motivates the design choices for implementation. Also the relationship between EM and MPM algorithm is shown in this chapter, which can help to understand the on-chip design a lot in a overall view.

In chapter 3, the overall hardware plan is presented. Two efficient on-chip structures named Pingpong and step structure are described. These are the key novel parallel hardware implementations. The detailed MPM algorithm implementation is described in this chapter.

Then the hardware memory interface design is described in chapter 4. Due to the large volume of data in 3D images, both internal FPGA and external on-board memory is necessary. This design minimizes external memory accesses.

Furthermore, the simulation and synthesis results are in Chapter 5. Compared with software implementation, the advantages of hardware implementation can be seen in both simulation and on-board segmentation results.

Finally, Chapter 6 concludes showing the advantages both in speed and cost of the 3D image segmentation in the FPGA platform.

2. 3D EM/MPM ALGORITHM

2.1 Introduction

For a given 3D image, the source image grey level information is considered a 3D volume of random variables, Y. For medical images, the model assumes that Y contains Gaussian noise due to the imaging process, plus the true underlying tissue characteristics. The segmentation result approximates the true tissues, denoted as X, without noise or distortion. This segmentation is also a 3D volume where there is assigned a class label corresponding to every pixel in the source 3D image. The class label is taken from a set of N labels. Described here is the optimization process by which we classify the pixels into the N labels.

The EM/MPM algorithm consists of two parts: Expectation-Maximization (EM) and Maximization of the Posterior Marginals (MPM). The EM algorithm finds the estimates for Gaussian mean and variance, while MPM classifies the pixels into N class labels, using the estimated parameters from EM. The basic structure of the image processing is a 3D neighborhood of pixels. In the 3D image research field, this forms a mathematical structure called a Markov Random Field (MRF). The MRF is useful because it guarantees local convergence in iterative algorithms which are based on it. The 3D 6-pixel neighborhood which we use is: right, left, above, below, front, and back around a center pixel.

A random class label is initialized into every pixel in X at the beginning of the segmentation process, and an evenly distributed vector of means and variances is used. Then, the estimate of X (the segmentation output, or class labeling) is formed by iterating several times through the 3D data. For MPM, convergence is achieved by choosing the class label that minimizes the expected value of the number of misclassified pixels, as proved in [3]. The probability density function (or likelihood function)

of a mixture of Gaussians, in which the random variable Y is dependent on X, is modeled in following Equation:

$$f_{Y|X}(y|x,\theta) = \prod_{s \in S} \frac{1}{\sqrt{2\pi\sigma_{x_s}^2}} \exp\left\{ -\frac{(y_s - \mu_{x_s})^2}{2\sigma_{x_s}^2} \right\} \tag{2.1}$$

θ is the vector of means and variances of each class (or tissue type), and the set S is the 3D volume of pixels with s denoting a single pixel.

Since we are assuming Bayesian dependence, we can use the p(x) to help solve this equation, resulting in Equation 2.2. Here, p(x) represents the tissue probable distribution in the 3D volume depending on the neighborhood class labels. This formulation will favor a class label for a center pixel that is similar to the largest number of neighboring class labels.

In order to get the approximation of this marginal conditional probability mass function at each pixel, a Gibbs sampler is used to generate a Markov chain X(t). After all the pixels have been processed through several iterations, EM uses class persistence from these iterations to estimate the new means and variances of the Gaussian models which is the input to MPM for the next iterative segmentation. After tens of EM iterations, the result of EM/MPM algorithm will converge to the highest probability segmentation.

2.2 3D Maximization of Posterior Marginals

The Equation 2.2 is used for MPM. The 3D pixel neighborhood is defined by the function $t(x_r, x_s)$, where x_s is the center pixel, and x_r are the nearest 6 pixels: up, down, left, right, front, and back.

The MPM optimization is used to segment images. This is accomplished by choosing a class label for every pixel in the estimate of X which can maximize the marginal probability mass functions in Equation 2.2.

$$p_{X_t|Y}(x|y,\theta) = \prod_{s\in S} \frac{1}{\sqrt{2\pi\sigma_{x_s}^2}} \exp\left\{ -\frac{(y_s - \mu_{x_s})^2}{2\sigma_{x_s}^2} - \sum_{[r,s]\in C} \beta t(x_s, x_r) \right\} \tag{2.2}$$

β : weighting factor for amount of spatial interaction

C : clique of X

y : source image

μ and σ : mean and variance for each class

The Gibbs sampler is the formulation used to create a Markov chain from the iterations. The Gibbs implementation in MPM is to choose a class label $x_s = k$, by using the uniform random variable ξ, compared to the neighborhood local posterior distribution $p(x_t)$ from Equation 2.2.

The Gibbs sampling becomes:

$$\begin{aligned} &\text{if } (\xi < p_1) \text{ then } x_t = \text{class label 1} \\ &\text{if } (p_1 < \xi < p_1 + p_2) \text{ then } x_t = \text{class label 2} \\ &\text{if } (p_1 + p_2 < \xi < p_1 + p_2 + p_3) \text{ then } x_t = \text{class label 3} \\ &\vdots \end{aligned} \tag{2.3}$$

MPM and EM have strong interrelationship, but MPM iterations are the majority of the computational processing, therefore the use of dedicated hardware is targeted to this algorithm. For each iteration of EM, the MPM iterates seven to ten times. MPM therefore is the target for parallelism and improved processing speed.

2.3 Expectation Maximization

EM is used to estimate parameter θ. For each iteration, two phases are implemented: the expectation step and the maximization step. First, the EM algorithm estimates the Gaussian hyper-parameters: θ as shown in the classic EM Equation 2.4.

$$Q(\theta, \widehat{\theta}(p-1)) = E_{Y,\widehat{\theta}(p-1)} \{\log f(y|x,\theta)\} + E_{Y,\widehat{\theta}(p-1)} \{\log p(x|\theta)\} \tag{2.4}$$

Then estimation of θ is obtained from maximizing $Q(\theta, \widehat{\theta}(p-1))$. In our 3D segmentation problem, the parameter vector, $\theta = (\mu_1, \sigma_1^2, \mu_2, \sigma_2^2, \cdots, \mu_N, \sigma_N^2)$, contains the statistics, means and variances, of the mixture probability density function, $f(Y|X)$ with the usual assumption of independent and identically distributed Gaussian random variables for each pixel.

3. HARDWARE IMPLEMENTATION

From the algorithm analysis, The EM/MPM algorithm diagram is shown in Figure 3.1. All the blocks in grey will be implemented in hardware; the blocks in red are external memory for storing source image Y and segmentation result Xt. The EM algorithm in blue will be implemented on the on-Board CPU which is named as Microblaze . It can be seen that each EM iteration calculates the mean and variance of each class based on segmentation results from m MPM loops. These new mean and variance are sent to MPM algorithm as input information for a new segmentation process at the next EM iteration.

3.1 Computational Cores

The MPM segmentation process is to minimize the exponential part for every class with respect to each pixel in Equation 3.1 which is named as $logpost(k)$ here:

$$logpost(k) = -\log \sigma_{x_s} - \frac{(y_s - \mu_{x_s})^2}{2\sigma_{x_s}^2} - \sum_{[r,s]\in C} \beta t(x_s, x_r) \tag{3.1}$$

So the computational block will classify every pixel, assigning it the class label k, based on smallest (magnitude) $logpost(k)$. This computational core is named *cal_cell*. It accomplishes the calculation of two important outputs: $logpost(k)$ and Xt_out. The Xt_out is the current estimated class for each pixel, which represents the current segmentation of the input. The Figure 3.2 is the core diagram.

The main ports for *cal_cell* are listed here, some other ports are from top level and not listed:

Input:

(1) $ena1$: calculation enable signal. When $ena1 = 1$, computational block works.

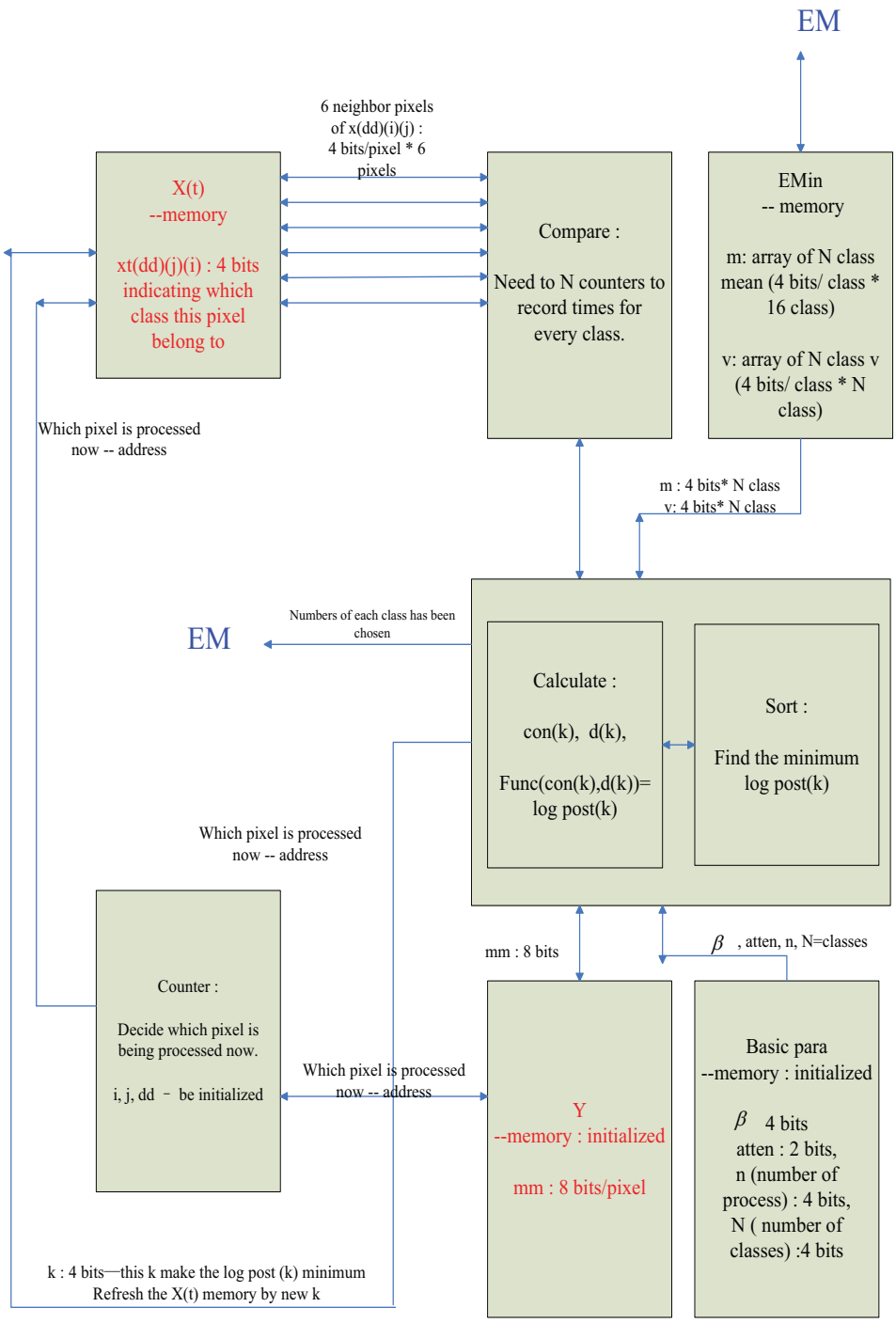

Fig. 3.1. Block Diagram for Algorithm

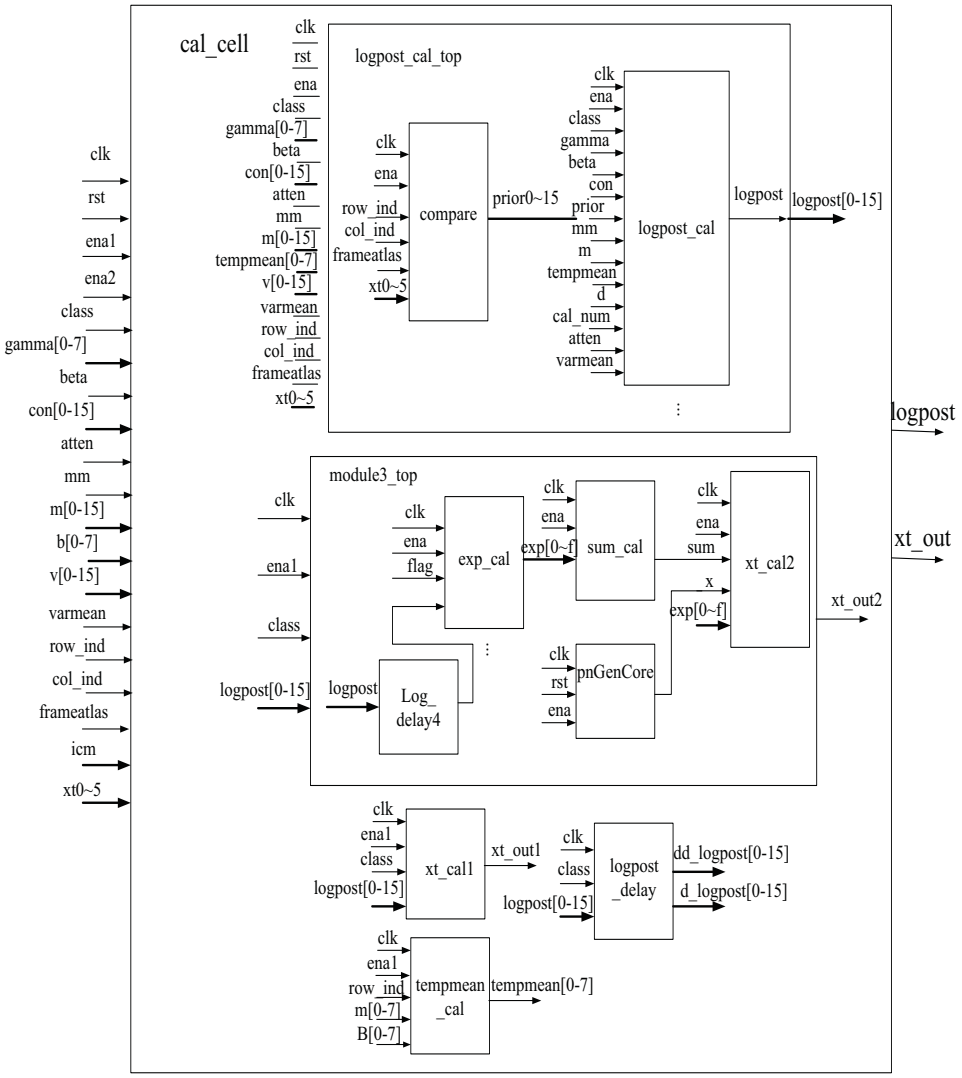

Fig. 3.2. Computational Core Diagram

(2) $ena2$: output enable signal. When $ena2 = 1$, outputs are validated.

(3) row_ind, $conl\_ind$,$frameatlas$: signals refer to row addresses, column addresses and slice addresses correspondingly. They are all outputs of ram_cell in which all the addresses are rearranged.

(4) $xt0$ - $xt5$: class labels of six neighboring pixels of current central pixel. They are read in from outside memory.

Output:

(1) *logpost*: output of this computational block. It illustrates cost function of every class.

(2) Xt_out: output of this computational block. It illustrates the class with this central pixel most likely belongs to.

Here are also several sub-block to compose this computational core:

(1) *tempmean_cal*: it computes the value of *tempmean*.

(2) *logpost_cal_top*: it is composed of two sub parts: *logpost_cal* (16 parallel blocks) and *compare*.

logpost_cal: it computes the value of *logpost*.

compare: it computers the value of 16 $prior(k)$ parallel. These $prior(k)$ are inputs for *logpost_cal*.

(3) Xt_cal1: when icm=0, sort to find smallest $logpost(k)$ among $logpost(0)$ to $logpost(k-1)$ and creates the Xt_out.

(4) *module3_top*: it is composed of several sub parts:

pnGenCore: IP core of Xilinx which generate 4 bits random number.

exp_cal: it calculates exponential function. It is hard to implement exponential calculation into hardware. We approximate this exponential calculation using four lines to fit the exponential curve in four different ranges.

Xt_cal2:when icm=1, calculate Xt_out.

As is shown in Figure 3.2 and comments above, the computational core computes the functions in the MPM algorithm. It classifies every pixel in Y according to its neighborhood in X, and the brightness value of Y according to a Gaussian model.

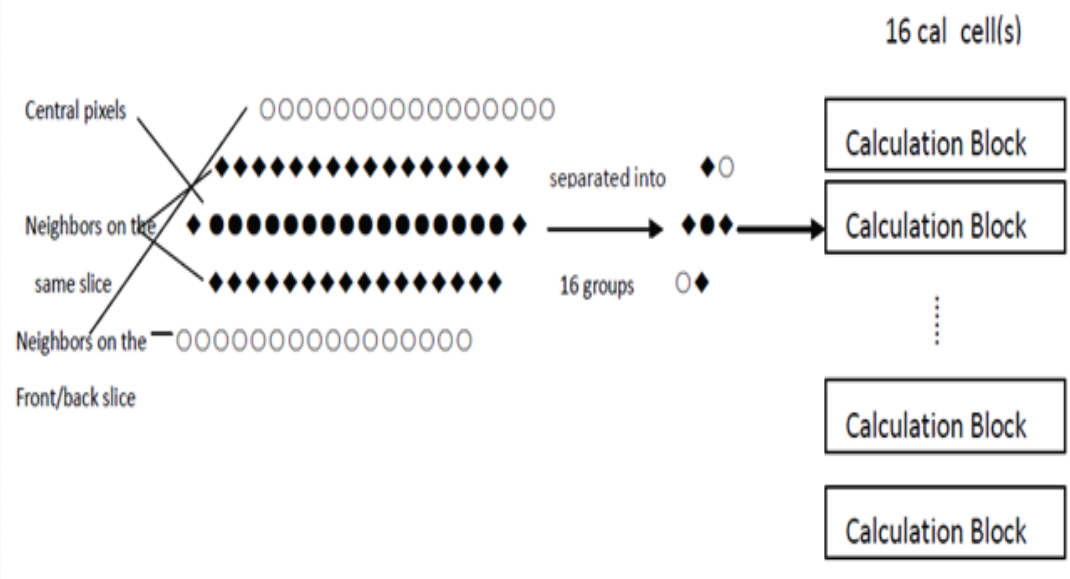

Fig. 3.3. Parallel Computational Cores Implementation

3.2 Parallel Cores Implementation

One of the most important advantages of hardware implementation is that several copies of the computational cores can be implemented in parallel. These multiple parallel cores improves the processing speed according to the number of cores. So considering the Xilinx chip hardware resources, there are sixteen computational cores implemented in this system.

Computational core *cal_cell* executes all the calculation functions for every pixel. Sixteen *cal_cell* are implemented in Xilinx FPGA parallel which process sixteen pixels at the same time. Figure 3.3 shows this parallel structure in the FPGA.

As is shown above, the data to classify sixteen pixels is read in at once. That is, Y of sixteen central pixels (black circles), and the current estimate, Xt, of their segmented neighbors from same slice and front/back slices (white circles and diamonds). After these are read in, the data are grouped into sixteen sets of 7 pixels each (corresponding to the sixteen black circles with the 6 neighbors in the diagram

above) and sent to multiple computational cores to calculate the new segmentation of those sixteen central pixels.

3.3 Two Important Hardware Architecture Design

3.3.1 Introduction

Our targeted algorithm MPM has three characteristics:

First of all, 6 neighboring pixels are involved in a certain central pixel's calculation. These 6 neighbors are in the same slice but in different rows from the central pixel, the right and left neighbors are in the same slice but indifferent column from the central pixel, and the front and back neighbors are in different slices but in same rows and columns from the central pixel. At the same time, every pixel is a neighbor for every other neighbor pixel.

Secondly, in order to converge the optimization, every pixel needs to be processed for seven to ten times in per EM iteration. In addition, every MPM iteration for one pixel involves this center pixel itself and its 6 neighbors refreshed from the last iteration.

Thirdly, there are a significant number of pixels in one 3D image and every pixel needs to be processed for seven to ten times. So we must pass a large quantity of data into or out from the chip. Therefore external memory is imperative. However, I/O speed is a bottleneck for most on-chip designs.

According to the characteristics mentioned above, three specific designs are created. The first one is called step structure. This step structure is aimed at the neighborhood calculation in MPM algorithm and partly decreases frequency of data exchange between internal and external memory. The second one is called PingPong structure. This design with step structure helps minimize the data exchange through I/O and saves the memory space both in internal and external memories.The third one is to take advantage of hardware to implement parallel processing by multiple

computational cores. The detailed implementation is related to multiple usages of RAMs.

3.3.2 PingPong Structure

According to the algorithm, 6 neighbors of a central pixel should be processed to the same MPM iteration level when they are used to refresh the central pixel to next level of MPM calculation. On the other hand, these 6 pixels will be used respectively in other groups of the neighborhood system. One choice is storing all the pixels in external memory and reading in and processing them one by one. After all the slices have been processed and written out, the 1st MPM calculation is finished for all pixels. The same process must be done for next several MPM loops. This method is obvious and intuitive. It can guarantee all the pixels are processed at the same level during every loop. But actually, this kind of process is technically inefficient. Because if MPM loops are processed for 7 times that means all the slices will be read in and written out for 7 times. However, as we know, frequent external RAM data access would result in low system processing speed.

In order to avoid frequent access to external memory, a new PingPong structure is created. This structure sets two groups of block RAMs - group RAMA and RAMB on chip. These RAMA and RAMB can hold 7 slices respectively. So my design is to treat these two RAMs as two PingPong players. The pixel data Xt on first 7 slices is read in from external memory and after it is processed for the first time,the result will be stored in RAMB. Then RAMB will be treated as the source memory, 7 slices data will be read out from RAMB and after second process, they will be stored in RAMA. Alternatively, RAMA will be used as source memory and RAMB will be the stored memory in third process. Then back and forth until all the 7 slices finishing 7 MPM loops. Any slice being processed for seven times will be considered as the final segmentation result in this EM iteration and sent out to external memory. Obviously,

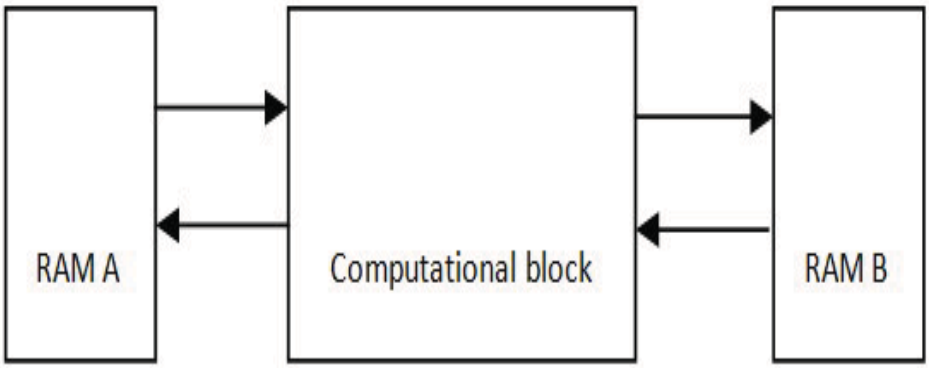

Fig. 3.4. PingPong Structure in RAMs

Pingpong structure guarantees that each slice will be read in and sent out once even they need to be processed for seven times in one MPM loop.

Figure 3.4 shows the PingPong structure used in this hardware implementation. Initially, the original data are read in and processed for the 1st time then stored in RAMB. At the 2nd time, data are read in from RAMB then processed once and then saved in RAMA. Now RAMA data is considered as the source data. The same procedure will be executed repeatedly until the 7th iteration is reached. This technique significantly reduces the data exchange between internal and external memory by keeping intermediate results on-chip. Finally, when the 7th iteration result is obtained it is written into the external memory. After the latency period of 7 iterations, the data transfer from external memory is continuous at one slice update per iteration. That is to say, any slices numbered S(n.1) to S(n.6)will be considered as intermediate results and transferred between inner memories. Any intermediate results will not be sent to external memory until S(n.7)is generated as the final result. Therefore we can process seven MPM segmentation iterations, matched to the external memory

bandwidth required of a single source file read, and a single segmented file write, plus a small amount of latency.

3.3.3 Step Structure

In this segmentation algorithm, the slice relevance and volume data exchange between internal and external memory make it difficult to accelerate processing speed. Slice relevance refers to the 3D neighborhood. While the central pixel is in process, the algorithm uses four pixels in the same slice and two pixels located in the previous slice and next slice. At the same time, the iterative nature of the algorithm means that a pixel that is processed for n iterations will be needed for the n+1 iteration. Using the previous segmentation result is a common property of optimization iterative convergence. This property is used in 3D image segmentation (and other 3D algorithms) and is difficult to parallelize in hardware because of the iterative nature and the neighborhood structure. Parallel cells in hardware help increase processing speed. But here, because of slice relevance, we can't group all the slices and assign them to parallel cells as separate tasks.

In order to use parallel calculation cells to speed up processing in hardware, a step structure is proposed here to deal with the slice relevance and iterative requirements. First of all, original slices are marked sequence starting from S(1) to S(n) along with the time (or spatial) horizon. At the very beginning, xt and y of first 2 slices are read in and sent to multiple calculation cells to process. After being processed one iteration, the first result slice is transferred to RAMB named as S(1.1). Here S(1.0) and S(2.0) will be held in inner rams. Then S(3.0) is read in and sent to multiple calculation cells to produce S(2.1) with S(1.0). This process continues until S(8.0) is read in. When S(7.1) is generated by S(6.0) and S(8.0) and stored in RAMB, it is considered first 7 slices are processed for the first time.Obviously, each slice should approach to S(n.7) when it has been processed and is ready to be sent to the external memory. Following RAM content figure shows RAM occupation at very beginning

Very beginning:		After first time: (from out memory to RAMB)	
RAM A	RAM B	RAM A	RAM B
0	0	0	$S_{1.1}$
0	0	0	$S_{2.1}$
0	0	0	$S_{3.1}$
0	0	0	$S_{4.1}$
0	0	0	$S_{5.1}$
0	0	0	$S_{6.1}$
0	0	0	$S_{7.1}$

Fig. 3.5. RAM Occupation at Beginning and after First Iteration

and when first 7 slices are processed once. Where S(n.0) is the original data of slice. S(n.m) is the n slice which has been processed for m times.

As is shown above, after being processed one iteration, the first result slice is transferred to RAMB named as S(n.1).The most important point is when S(8.0) is read in and S(7.1) is generated, reading new xt from outside memory is stopped. For the next iteration, RAMB is considered as source memory. S(2.1) is used to generate S(1.2) and this S(1.2) is stored in RAMA, S(1.1) and S(3.1) are used to generate S(2.2) and this S(2.2) is stored in RAMA, then in the final step of second iteration, S(5.1) and S(7.1) are used to generated S(6.2) and this S(6.2) is stored in RAMA. Basically, the strategy is using PingPong structure of two RAMs to hold all the intermediate results in inner RAMs. Following figure shows the data transformation between inner RAMs for first 7 slices.

Very beginning:		After first time: (from out memory to RAMB)		After second time: (from RAMB to RAMA)	
RAM A	RAM B	RAM A	RAM B	RAM A	RAM B
0	0	0	$S_{1.1}$	$S_{1.2}$	$S_{1.1}$
0	0	0	$S_{2.1}$	$S_{2.2}$	$S_{2.1}$
0	0	0	$S_{3.1}$	$S_{3.2}$	$S_{3.1}$
0	0	0	$S_{4.1}$	$S_{4.2}$	$S_{4.1}$
0	0	0	$S_{5.1}$	$S_{5.2}$	$S_{5.1}$
0	0	0	$S_{6.1}$	$S_{6.2}$	$S_{6.1}$
0	0	0	$S_{7.1}$	0	$S_{7.1}$
After third time: (from RAMA to RAMB)		After fourth time: (from RAMB to RAMA)		After third time: (from RAMA to RAMB)	
RAM A	RAM B	RAM A	RAM B	RAM A	RAM B
$S_{1.2}$	$S_{1.3}$	$S_{1.4}$	$S_{1.3}$	$S_{1.4}$	$S_{1.5}$
$S_{2.2}$	$S_{2.3}$	$S_{2.4}$	$S_{2.3}$	$S_{2.4}$	$S_{2.5}$
$S_{3.2}$	$S_{3.3}$	$S_{3.4}$	$S_{3.3}$	$S_{3.4}$	$S_{3.5}$
$S_{4.2}$	$S_{4.3}$	$S_{4.4}$	$S_{4.3}$	$S_{4.4}$	$S_{4.3}$
$S_{5.2}$	$S_{5.3}$	$S_{5.2}$	$S_{5.3}$	$S_{5.2}$	$S_{5.3}$
$S_{6.2}$	$S_{6.1}$	$S_{6.2}$	$S_{6.1}$	$S_{6.2}$	$S_{6.1}$
0	$S_{7.1}$	0	$S_{7.1}$	0	$S_{7.1}$
After sixth time: (from RAMB to RAMA)		After seventh time: (from RAMA to RAMB)			
RAM A	RAM B	RAM A	RAM B		
$S_{1.6}$	$S_{1.5}$	$S_{1.6}$	$S_{1.7}$		
$S_{2.6}$	$S_{2.5}$	$S_{2.6}$	$S_{2.5}$		
$S_{3.4}$	$S_{3.5}$	$S_{3.4}$	$S_{3.5}$		
$S_{4.4}$	$S_{4.3}$	$S_{4.4}$	$S_{4.3}$		
$S_{5.2}$	$S_{5.3}$	$S_{5.2}$	$S_{5.3}$		
$S_{6.2}$	$S_{6.1}$	$S_{6.2}$	$S_{6.1}$		
0	$S_{7.1}$	0	$S_{7.1}$		

Fig. 3.6. Transformation Records during Seven Times Process

m = 7	$S_{1.7}$							
m = 6	$S_{1.6}$	$S_{2.6}$						
m = 5	$S_{1.5}$	$S_{2.5}$	$S_{3.5}$					
m = 4	$S_{1.4}$	$S_{2.4}$	$S_{3.4}$	$S_{4.4}$				
m = 3	$S_{1.3}$	$S_{2.3}$	$S_{3.3}$	$S_{4.3}$	$S_{5.3}$			
m = 2	$S_{1.2}$	$S_{2.2}$	$S_{3.2}$	$S_{4.2}$	$S_{5.2}$	$S_{6.2}$		
m = 1	$S_{1.1}$	$S_{2.1}$	$S_{3.1}$	$S_{4.1}$	$S_{5.1}$	$S_{6.1}$	$S_{7.1}$	
RD_IN	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8

Fig. 3.7. Transformation Results after First Slice being Produced

As is shown in Figure 3.6, 7 slices held in inner RAMs are processed to different levels because of slices relevance. The remaining slices have only been processed partially (since we need new slice data). So the maintenance of the slice relevance is necessary: because the front and back slice which have been processed for m times are needed to refresh the central slice for m+1 times, so S(n.m) then can be processed only if S(n-1.m-1)and S(n+1.m-1) are available.

Next, this different extent processeing is used to solve the iterative and neighborhood structure issues in this particular algorithm. Figure 3.7 shows this different extent of processing in a horizontal way which is like a step. Where: m: MPM iteration number.

As is shown in Figure 3.7, from bottom to up and from left to right, 1st slice is processed for 7 times and ready to be sent out to external memory. 2nd slice is processed for 6 times, 3rd slice is processed for 5 times and ect. That is to say, from very beginning, there are 28 slice-processes for first 7 slices until 1st slice is processed

After seventh time: (from RAMA to RAMB)		After $S_{7.1}$ is sent out and $S_{8.1}$ being produced:	
RAM A	RAM B	RAM A	RAM B
$S_{1.6}$	$S_{1.7}$	$S_{1.6}$	$S_{8.1}$
$S_{2.6}$	$S_{2.5}$	$S_{2.6}$	$S_{2.5}$
$S_{3.4}$	$S_{3.5}$	$S_{3.4}$	$S_{3.5}$
$S_{4.4}$	$S_{4.3}$	$S_{4.4}$	$S_{4.3}$
$S_{5.2}$	$S_{5.3}$	$S_{5.2}$	$S_{5.3}$
$S_{6.2}$	$S_{6.1}$	$S_{6.2}$	$S_{6.1}$
0	$S_{7.1}$	0	$S_{7.1}$

Fig. 3.8. Inner RAM Occupation after S(8.1) being Produced

to the end. After 1st slice being finished and sent out, 9th slice is read in and produces S(8.1) with S(7.0). Then this S(8.1) will be send to RAMB to make up valid left by S(1.7). Let's continue think about RAM occupation. Figure 3.8 shows the result after S(8.1) being produced and next iteration being processed to every slice in inner RAMs.

After S(8.1) making up valid of S(1.7), RAMA and RAMB are considered as source ram alternatively per slice. S(7.2) is produced by S(8.1) and S(6.1) and stored in RAMA; S(7.2) is produced by S(8.1) and S(6.1) and stored in RAMA S(3.6) is produced by S(2.5) and S(4.5) and stored in RAMA; S(2.7) is produced by S(1.6) and S(3.6) and sent out. Following figure shows next step when S(2.7) is ready.

When S(2.7) is sent out, S(9.1) is produced by S(8.0) and S(10.0) then make up valid of S(2.7). Figure 3.9 shows changes in RAMA and RAMB in following few steps.

After seventh time: (from RAMA to RAMB)		After $S_{7.1}$ is sent out and $S_{8.1}$ being produced:		When $S_{2.7}$ is ready to be sent out	
RAMA	RAMB	RAMA	RAMB	RAMA	RAMB
$S_{1.6}$	$S_{1.7}$	$S_{1.6}$	$S_{8.1}$	$S_{1.6}$	$S_{8.1}$
$S_{2.6}$	$S_{2.5}$	$S_{2.6}$	$S_{2.5}$	$S_{2.6}$	$S_{2.7}$
$S_{3.4}$	$S_{3.5}$	$S_{3.4}$	$S_{3.5}$	$S_{3.6}$	$S_{3.5}$
$S_{4.4}$	$S_{4.3}$	$S_{4.4}$	$S_{4.3}$	$S_{4.4}$	$S_{4.5}$
$S_{5.2}$	$S_{5.3}$	$S_{5.2}$	$S_{5.3}$	$S_{5.4}$	$S_{5.3}$
$S_{6.2}$	$S_{6.1}$	$S_{6.2}$	$S_{6.1}$	$S_{6.2}$	$S_{6.3}$
0	$S_{7.1}$	0	$S_{7.1}$	$S_{7.2}$	$S_{7.1}$

Fig. 3.9. Inner RAM Results when S(2.7) is Produced

After $S_{7.1}$ is sent out and $S_{8.1}$ being produced:		When $S_{2.7}$ is ready to be sent out		When $S_{9.1}$ is produced and stored in RAMB	
RAMA	RAMB	RAMA	RAMB	RAM A	RAM B
$S_{1.6}$	$S_{8.1}$	$S_{1.6}$	$S_{8.1}$	$S_{1.6}$	$S_{8.1}$
$S_{2.6}$	$S_{2.5}$	$S_{2.6}$	$S_{2.7}$	$S_{2.6}$	$S_{9.1}$
$S_{3.4}$	$S_{3.5}$	$S_{3.6}$	$S_{3.5}$	$S_{3.6}$	$S_{3.5}$
$S_{4.4}$	$S_{4.3}$	$S_{4.4}$	$S_{4.5}$	$S_{4.4}$	$S_{4.5}$
$S_{5.2}$	$S_{5.3}$	$S_{5.4}$	$S_{5.3}$	$S_{5.4}$	$S_{5.3}$
$S_{6.2}$	$S_{6.1}$	$S_{6.2}$	$S_{6.3}$	$S_{6.2}$	$S_{6.3}$
0	$S_{7.1}$	$S_{7.2}$	$S_{7.1}$	$S_{7.2}$	$S_{7.1}$
When $S_{3.7}$ is ready to be sent out		When $S_{10.1}$ is produced and stored in RAMB		When $S_{4.7}$ is ready to be sent out	
RAMA	RAMB	RAMA	RAMB	RAM A	RAM B
$S_{8.2}$	$S_{8.1}$	$S_{8.2}$	$S_{8.1}$	$S_{8.2}$	$S_{8.1}$
$S_{2.6}$	$S_{9.1}$	$S_{2.6}$	$S_{9.1}$	$S_{9.2}$	$S_{9.1}$
$S_{3.6}$	$S_{3.7}$	$S_{3.6}$	$S_{10.1}$	$S_{3.6}$	$S_{10.1}$
$S_{4.6}$	$S_{4.5}$	$S_{4.6}$	$S_{4.5}$	$S_{4.6}$	$S_{4.7}$
$S_{5.4}$	$S_{5.5}$	$S_{5.4}$	$S_{5.5}$	$S_{5.6}$	$S_{5.5}$
$S_{6.4}$	$S_{6.3}$	$S_{6.4}$	$S_{6.3}$	$S_{6.4}$	$S_{6.5}$
$S_{7.2}$	$S_{7.3}$	$S_{7.2}$	$S_{7.3}$	$S_{7.4}$	$S_{7.3}$

Fig. 3.10. Inner RAM Results in Following Steps

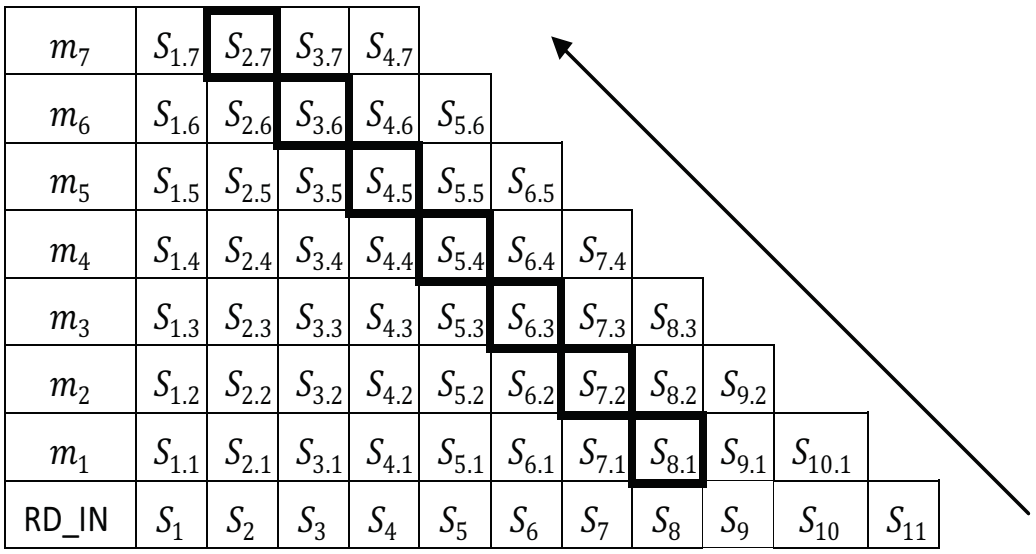

Fig. 3.11. Following Steps after First Slice being Sent Out

It is obvious that every slice in inner RAM is refreshed to a successively deeper iteration number (m). Any slice refreshed 7 times is considered as finished and sent out. Figure 3.10 shows the steps after S(1.7) was sent out in a horizontal way which is like a step.

As is shown above, When the 1st slice is sent out what remains are the other intermediate slices, the processes will continue. The 2nd slice will be ready after S(8.1) to S(3.6) (in bold cells in Figure 3.11, from bottom to up) are produced. After filling in the diagonal in above figure continuously, then another 6 intermediate results and one final slice segmentation are produced, and every slice is considered ready to be sent out to external memory. Any intermediate results from T(1) to T(7) are kept in inner memory. This process of 7 times loops for every slice can be done with one time read-in and one time write-out, which decreases the frequency of access to external memory and avoids unnecessary I/O transfers.

4. EXTERNAL MEMORY INTERFACE DESIGN

4.1 Introduction

Memory arrangement and interface design is very important for this project. On one hand, almost every 3D image involves a huge amount of data which needs to be exchanged between external and internal memory. On the other hand, I/O speed is always a bottleneck for the processing speed of hardware. So an efficient memory arrangement and an effective memory interface controller are critical for system speed.

From the algorithm analysis above, current segmentation Xt and original source image Y are volume inputs to this system. Refreshed segmentation Xt is volume output data of this system. According to the system requirements, every Xt is assigned with 4 bits (maximum 16 class labels) and every Y is assigned 8 bit greyscale. Therefore, 12 bits are needed to read in and 4 bits write out per pixel.

4.2 Memory Arrangement

Because of the large amount of data is involved in this algorithm, external memory is necessary. Using our two novel hardware designs, an inner memory which can hold several slices of Xt is also needed.

4.2.1 External Memory (DDR3) Analysis

From analysis of algorithm, y of central pixel and xt of its six neighbors should be sent to calculation blocks at same time. So the source image volume Y and the current segmentation field volume Xt are two read in data flows. In order to read in data more efficiently, Y of S(n) is stored with Xt of S(n+1) in the same address.For

Y of S_1	Xt of S_2
Y of S_2	Xt of S_3
Y of S_3	Xt of S_4
.....	
.....	
.....	
Y of S_{126}	Xt of S_{127}
Y of S_{127}	Xt of S_{128}
Y of S_{128}	Xt of S_1

Fig. 4.1. External Memory Storage Arrangement

example, the 3D image is 128*128*128, the external memory storage arrangement is as follows.

At very beginning, Xt of S(1) is generated randomly and stored in inner memory on chip. Y of S(1) and Xt of S(2) are read in. Xt of S(1), Y of S(1) and Xt of S(2) are sent into sixteen calculation blocks to get Xt of S(1.1) which is sent and stored in RAMB. Then Y of S(2) and Xt of S(3) are read in. Y of S(2), Xt of S(1), Xt of S(2) and Xt of S(3) are sent into sixteen calculation blocks to get Xt of S(2.1). All the slices are processed according to the order listed in Figure 3.11. After S(1.7) which is refreshed Xt of S(1) is ready, it will be sent out to external memory in the

same address of Y of S(128). After S(2.7) which is refreshed Xt of S(2) is ready, it will be sent out to external memory in the same address of Y of S(1). There are two advantages to store Y and Xt in this order. First, mixing Y and Xt in same address can avoid frequent changes of addresses of DDR3. This is because staggered Y and Xt sends the whole data into calculation blocks together at one time. Here we avoid changing the addressing randomly because it will cause latency in reading and writing process of DDR3. Second, Y is kept one slice ahead of Xt. This arrangement makes pipelining more efficient. Both Y of central slice and Xt of back slice will be used in refreshing central pixels. When they are read in at the same time and Xt of front slice is already kept in inner memory so read-in Y doesn't need to wait for Xt of back slice.

4.2.2 Inner Memory Analysis

In order to avoid high frequency external memory accesses, some inner memory are designed on chip. Here are four parts of inner memory as follows:

(1) Inner memory for step structure and PingPong structure:

(2) slices for Xt(Step structure) * 2 (PingPong structure). As is shown in Figure 3.3, in order to make computation pipeline more efficient, neighbors of central pixels should be sent to computation blocks at the same time. So here the PingPong structure RAMs are doubled in order to read in all neighbors in central slice during one clock cycle:

(3) slices for Xt (Step structure) * 2 and (PingPong structure) * 2

(4) There will be a buffer for 2 slices of Y. To maximize the pipeline calculation we designed so that the computation blocks never stop. However, we do not want the read-in process stop when Xt is written out. So this buffer holds Y to keep the computation pipeline working when Xt is written out.

When Xt of S(n-1) and S(n) are read in,they are kept in two RAMs until S(n+1) is read in. Then S(n) is processed and S(n-1) will be replaced by S(n+1). These two

little RAMs which hold two slices of Xt is like a ball-server for PingPong structure. So other 2 slices for Xt are added here.

4.2.3 Interface Controller Design

Between external memory and inner memory, there is an interface to rearrange Xt and Y data to fit the computational cores. At the same time, this interface should control slice storage of Xt and Y. Figure 4.2 is shows this control process.

The calculation system and memory interface is connected as as in Figure 4.3.

The detailed data controller working process is like Figure 4.2. At very beginning, Xt of S(1) is generated randomly and stored in RAM S(n-1). Xt of S(2) and Y of S(1) are read in together and sent to the Pixels-Grouper. Then S(1.1) is produced and stored to RAMB. Next, Xt of S(3) and Y of S(2) are read in together. These Xt of S(3) and Y of S(2) are sent to the Pixels Grouper with Xt of S(2) and Xt of S(1). And then Xt of S(3) will replace S(1) to wait for the next Xt read in.

When the process is approaching, S(n,6), the read-in Y of S(n,6) is stored in those two Y buffer RAMs. It is obviously that in next step, Xt of S(n,7) is written out and read-in process will stop. So at this time, computation cores will use the stored Y in 6 iteration of every slice to keep pipelining.

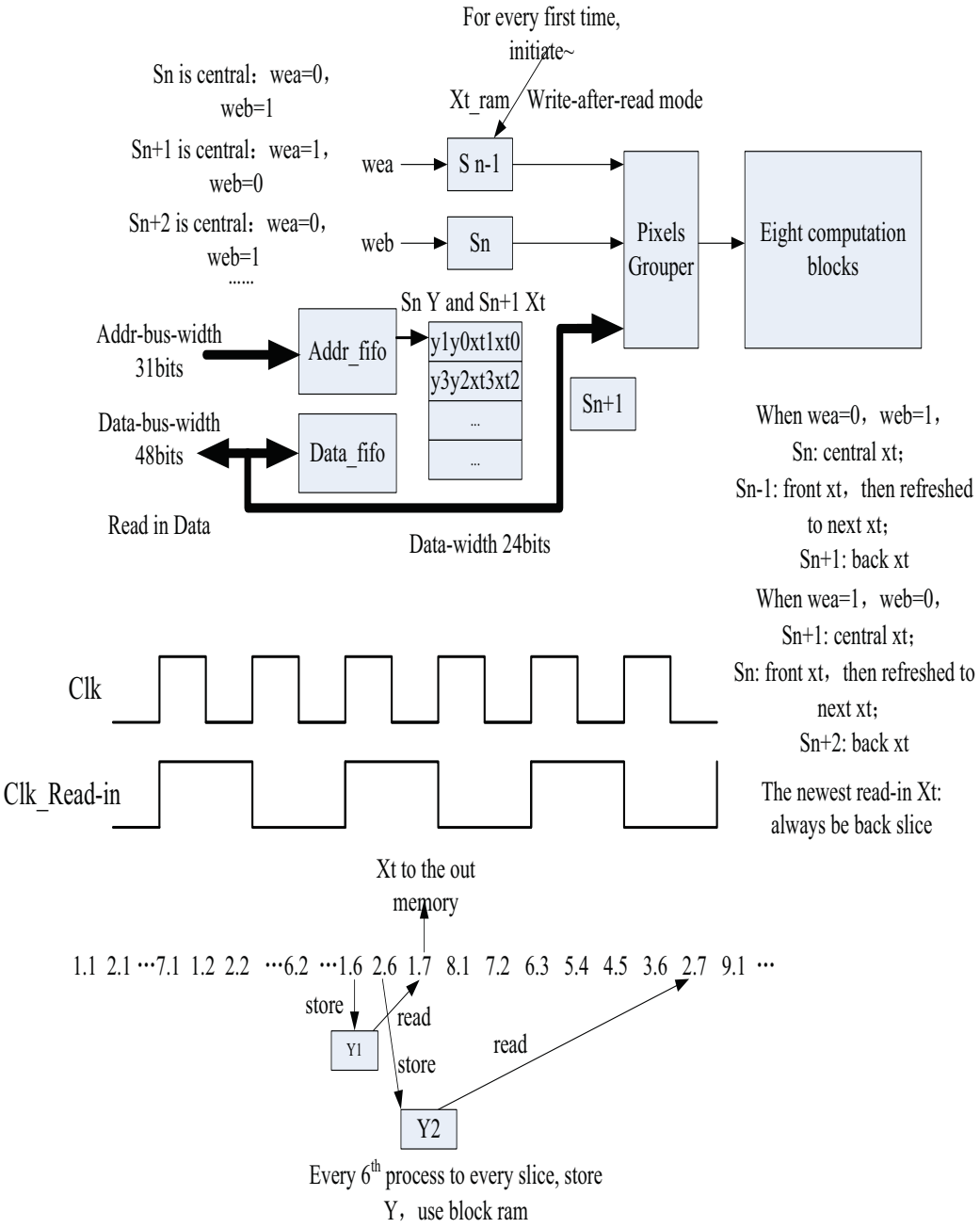

Fig. 4.2. Memory Interface Design and Data Rearrangement

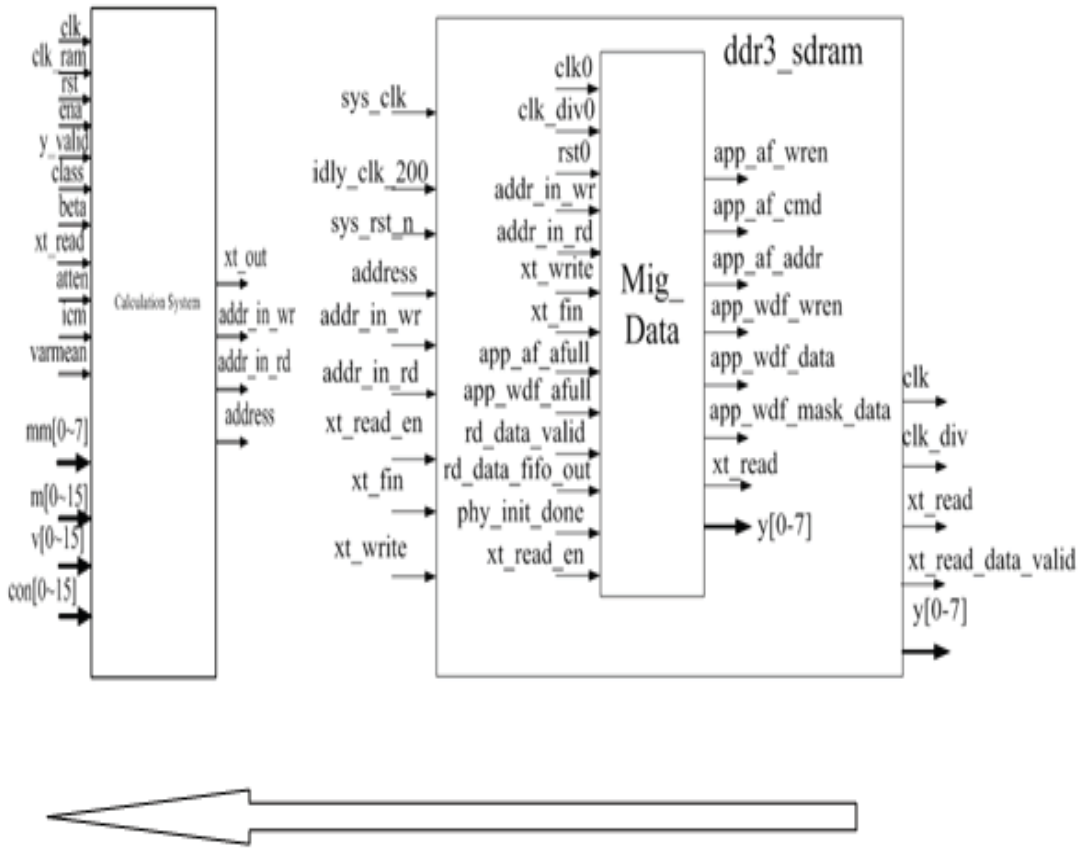

Fig. 4.3. Calculation Part and Memory Interface Connection

5. RESULTS: SYNTHESIS AND SIMULATION

5.1 Hardware Synthesis Resource Analysis

Our simulation work is based on Xilinx Virtex 6vLX240Tff1156-2. The basic on-board resource is shown highlighted in the bold box in Figure 5.1.

With 16 computation cores and external memory interface implemented on chip, the resource usage in synthesis report is shown in Figure 5.2.

It is shown that all the resource usage is under 70%, which makes on-chip implementation achievable and scalable.

Device	Logic Cells	Configurable Logic Blocks (CLBs)		DSP48E1 Slices[2]	Block RAM Blocks			MMCMs[4]	Interface Blocks for PCI Express	Ethernet MACs[5]	Maximum Transceivers		Total I/O Banks[6]	Max User I/O[7]
		Slices[1]	Max Distributed RAM (Kb)		18 Kb[3]	36 Kb	Max (Kb)				GTX	GTH		
XC6VLX75T	74,496	11,640	1,045	288	312	156	5,616	6	1	4	12	0	9	360
XC6VLX130T	128,000	20,000	1,740	480	528	264	9,504	10	2	4	20	0	15	600
XC6VLX195T	199,680	31,200	3,040	640	688	344	12,384	10	2	4	20	0	15	600
XC6VLX240T	241,152	37,680	3,650	768	832	416	14,976	12	2	4	24	0	18	720
XC6VLX365T	364,032	56,880	4,130	576	832	416	14,976	12	2	4	24	0	18	720
XC6VLX550T	549,888	85,920	6,200	864	1,264	632	22,752	18	2	4	36	0	30	1200
XC6VLX760	758,784	118,560	8,280	864	1,440	720	25,920	18	0	0	0	0	30	1200
XC6VSX315T	314,880	49,200	5,090	1,344	1,408	704	25,344	12	2	4	24	0	18	720
XC6VSX475T	476,160	74,400	7,640	2,016	2,128	1,064	38,304	18	2	4	36	0	21	840
XC6VHX250T	251,904	39,360	3,040	576	1,008	504	18,144	12	4	4	48	0	8	320
XC6VHX255T	253,440	39,600	3,050	576	1,032	516	18,576	12	2	2	24	24	12	480
XC6VHX380T	382,464	59,760	4,570	864	1,536	768	27,648	18	4	4	48	24	18	720
XC6VHX565T	566,784	88,560	6,370	864	1,824	912	32,832	18	4	4	48	24	18	720

Fig. 5.1. Xilinx Virtex 6vLX240Tff1156-2 on-chip Resource

Selected Device : 6vlx240tff1156-2

Slice Logic Utilization:
Number of Slice Registers: 43043 out of 301440 14%
Number of Slice LUTs: 51005 out of 150720 33%
Number used as Logic: 43587 out of 150720 28%
Number used as Memory: 7418 out of 58400 12%
Number used as RAM: 372
Number used as SRL: 7046

Slice Logic Distribution:
Number of LUT Flip Flop pairs used: 59366
Number with an unused Flip Flop: 16323 out of 59366 27%
Number with an unused LUT: 8361 out of 59366 14%
Number of fully used LUT-FF pairs: 34682 out of 59366 58%
Number of unique control sets: 450

IO Utilization:
Number of IOs: 385
Number of bonded IOBs: 385 out of 600 64%

Specific Feature Utilization:
Number of Block RAM/FIFO: 4 out of 416 0%
Number using FIFO only: 4
Number of BUFG/BUFGCTRLs: 6 out of 32 18%

Fig. 5.2. Resource Usage Report

5.2 Simulation Results Analysis

Our test case for simulation is a 128*128*128 3D medical image. The Y data and Xt data are 8 bits and 4 bits respectively. For the simulation case, we just show the first slice, 7th MPM iteration result.

The simulation work based on Modelsim SE6.2 using Xilinx Vertex6lx240t FPGA. The read in and write out clock for external DDR3 memory is set at 200MHz. The clock for the computational core is 100MHz. Two requirements should be considered when choosing the clock frequency. First is the limitation from I/O interface. For this Xilinx Virtex6 development board, the external memory access clock limitation is 333MHz. So the memory interface clock for accessing external memory should be below 333MHz. Another requirement is that the computational clock should be less than half of the external memory clock to guarantee the continuity of computational pipeline process. Due to the DDR3 timing, there is half a clock period to read-in data from external memory and half a clock to write out the result to external memory.

From Equation 3.1, the input data are: original image information Y, prior segmentation Xt for each pixel and class means and variance for each class. In the simulation all the data are changed to hex format and saved in a text file.

When simulation starts, the first task is to initialize all Y and Xt to external memory. Figure 5.3 shows that after all the Y and Xt are available in external DDR3 memory, the read in process starts, this is achieved in about 66μs.

Upon being read-in, the Y and Xt are sent to calculation cores cal_cell to process. Then in 88.25μs, the renewed first iteration Xt, which is also the segmentation result for first 16 pixels, is sent out. After that, the computational process is pipelined and the segmentation results then will come out one pixel per computational clock circle.

It can be seen from the address accumulation signal, the segmentation for the first slice is finished in 396μs. This is compared to our calculation by hand of is about 376μs. The difference is coming from the external memory address delay during DDR3 page transition. From the simulation data, we can conclude that, for this 128*128*128

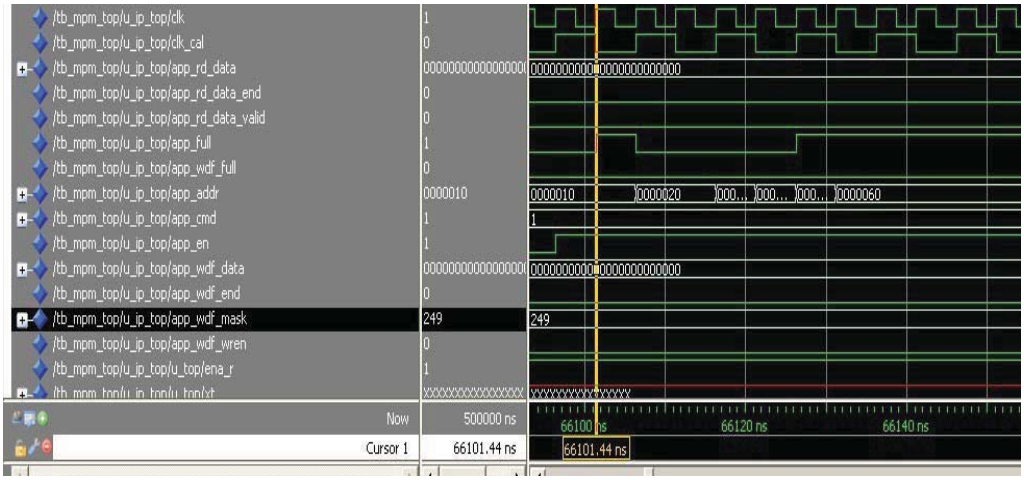

Fig. 5.3. Read-in Process Starts

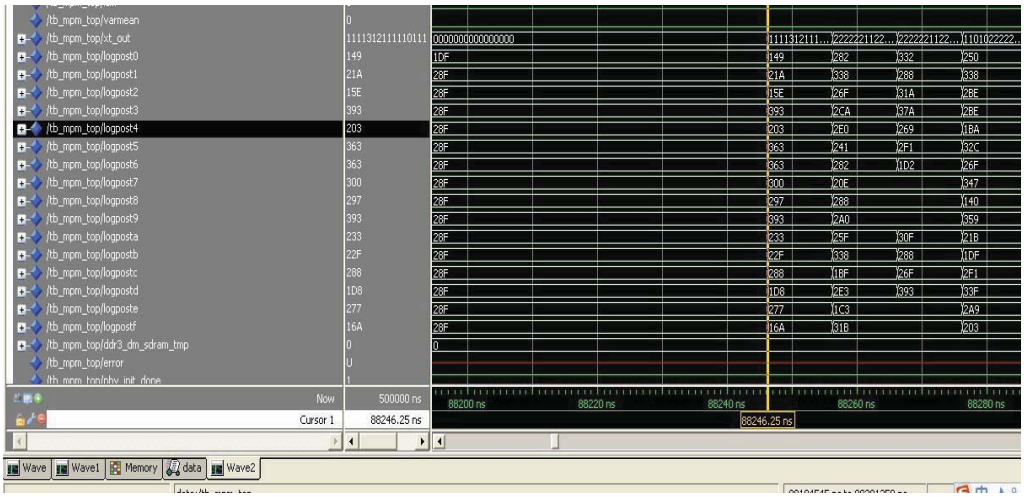

Fig. 5.4. First Xt Comes Out

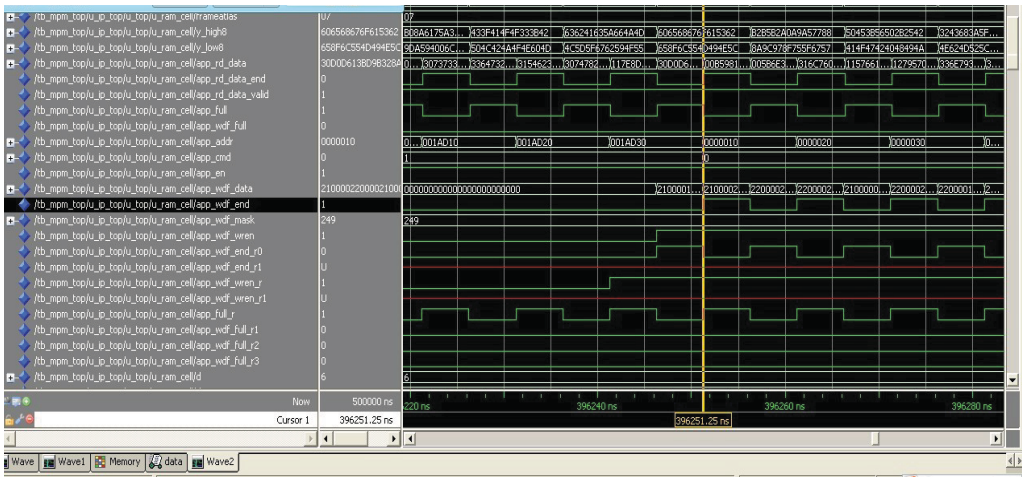

Fig. 5.5. First Slice Calculation Finishes

volume 3D image with 7 MPM iterations complete, there is 0.3ms latency followed by each subsequent slice available every 0.072ms. Total time for the complete volume with 7 MPM iterations is 9.5ms. For normal EM convergence, we would have 20 of these cycles, making the total segmentation for this size volume approximately 200ms. Scaling up to a typical size of medical image, 512*512*512, we would have about 12 seconds (0.2 minutes) of processing time with the hardware acceleration, compared to 25 minutes on a quad core PC, thus we have achieved a 100 times acceleration.

From the result, we can see that there are still a timing difference between our expectation and simulation result. This difference comes from the detailed external memory in read-in and write-out processes. To further improve this, we can increase the fifo size slightly for Y or decrease slightly the computational clock frequency.

After first slice is sent out, the result can be seen under memory tab in Modelsim platform as shown in Figure 5.6. The contents are the final segmentation result for slice 1 using current mean and variance.

We can pull out the result to a text file and using IMAGEJ software to export image, the result is shown in Figure 5.7. Figure 5.8 is first iteration result from the

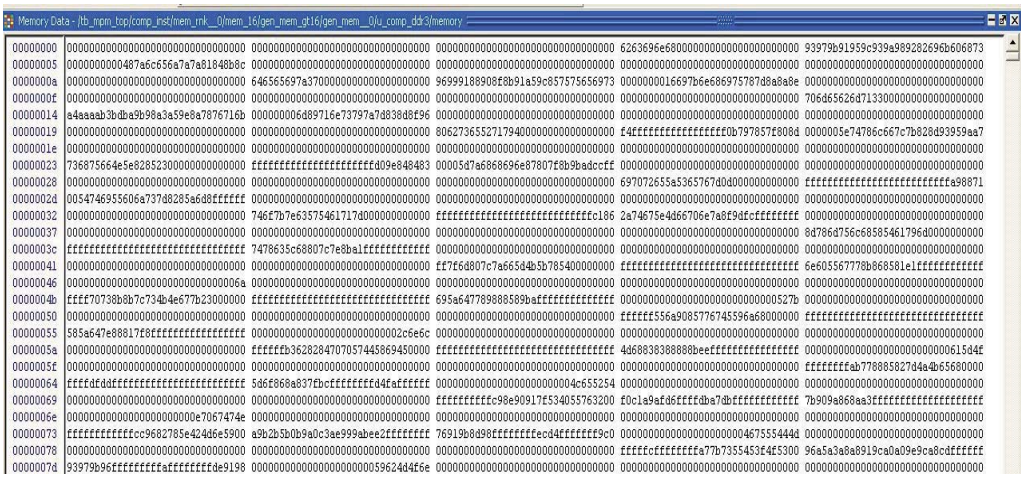

Fig. 5.6. Simulation Result for First Slice in External Memory

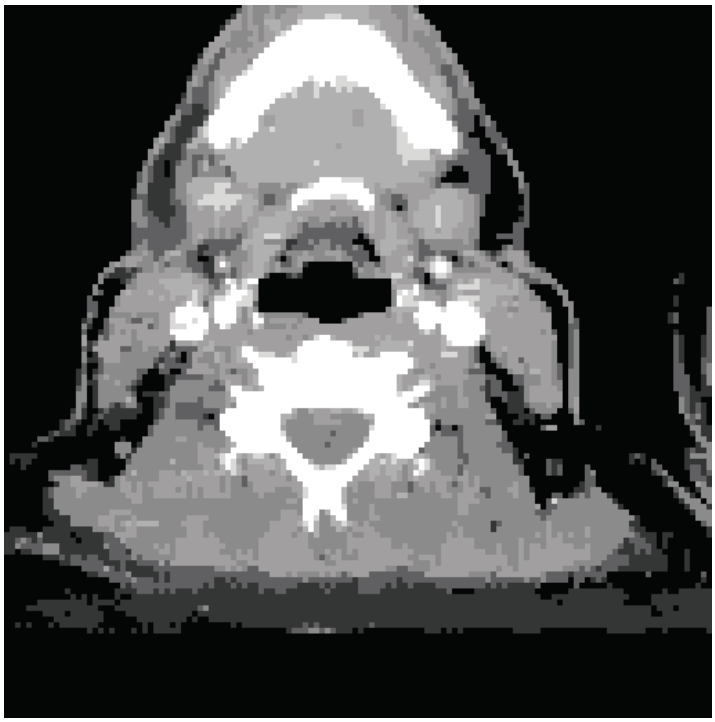

Fig. 5.7. First Iteration Result of Xilinx Hardware Segmentation

standard desktop computer using software to process the same data. We can conclude from above images that hardware and software results are almost the same. Based on the simulation result, we compare processing time between our implementation on hardware and on standard desktop computer executing software. The result is shown in Figure 5.9. It can be concluded that the hardware advantage is 100 times the processing speed. Also, the processing time is compared with the referenced hardware implementations based on different 3D segmentation algorithms. The result is shown in Figure 5.10. Taking the published data from reference [10], we scaled down the time to 31.35ms, in order to match the 128x128x128 size. It can be seen that our hardware implementation based on EM/MPM algorithm makes a significant acceleration.

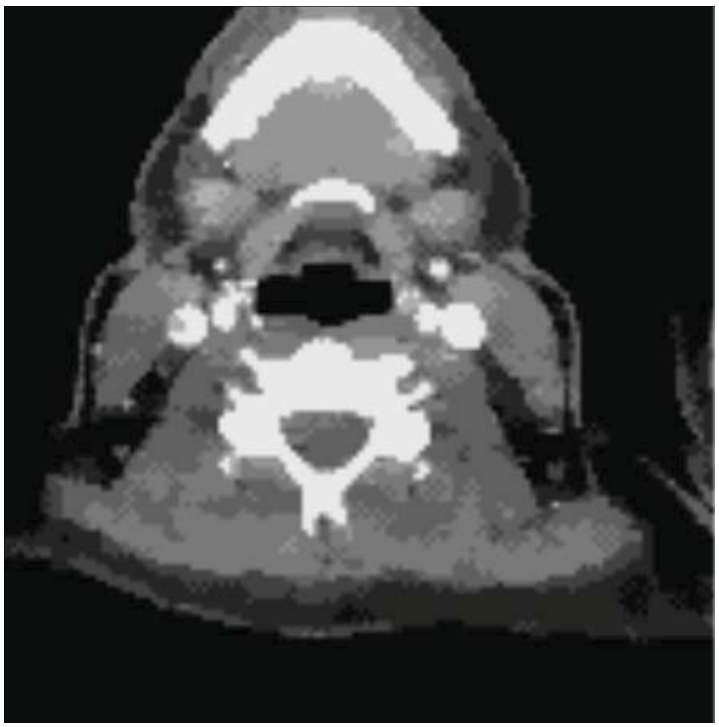

Fig. 5.8. First Iteration Result of PC Software Segmentation

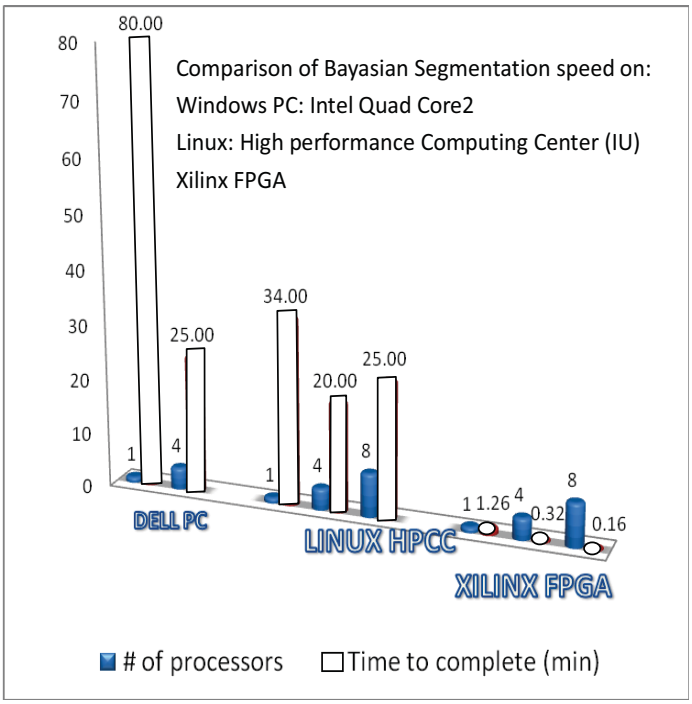

Fig. 5.9. Hardware Processing Speed Comparison with Software

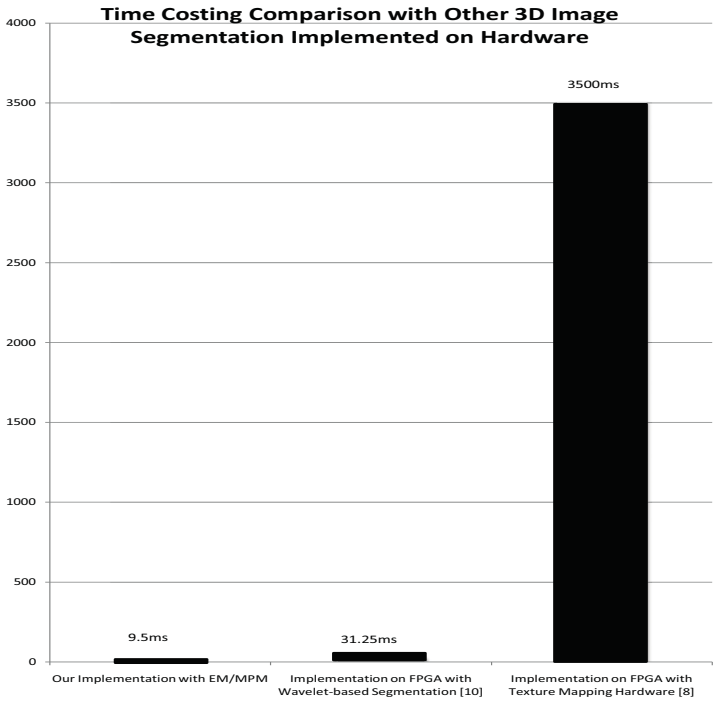

Fig. 5.10. Processing Speed Comparison with Literature Hardware Implementations

6. CONCLUSION AND FUTURE RESEARCH

6.1 Conclusion

In this thesis we have proposed a new hardware implementation design for EM/MPM algorithm based on Xilinx Virtex6 development board. This new hardware structure is designed to accelerate whole image segmentation process compared to software. Through implementing multiple computational cores on chip and designing a good I/O interface to avoid I/O speed limitations, it has been proved that our hardware design does speed up the whole 3D image segmentation process by at least 100 times and is an improvement from the literature by more than 3 times.

In Chapter 1, we have reviewed several image segmentation algorithms. Specifically we compared algorithms applied on 3D image segmentation and we chose EM/MPM algorithm to implement in hardware because of the good performance, especially in noise. Also, we showed that the hardware implementation has several advantages compared to software solution both from processing speed and resource cost aspects.

In Chapter 2, we briefly introduced the concept of EM/MPM algorithm and pointed out that MPM will be the main part on hardware based on the nature of algorithm itself.

In Chapter 3, we have discussed the characteristics of MPM algorithm and based on these characteristics, PingPong Structure and Step Structure are proposed. PingPong structure targets on-chip iterative processing, and Step structure reduces the I/O interface between on-chip and external memories. Multiple parallel computational cores are implemented on hardware which process the image concurrently and accelerate the processing speed significantly.

In Chapter 4, we have proposed a new I/O interface design which can help reduce external memory access with the step structure. I/O interface speed limitation is always the bottleneck for speeding up hardware processing speed, especially for large data volume involving processing. Our original design successfully solved this problem and made the data read-in and write-out process excute smoothly without stopping the pipelined computational processes on chip.

In Chapter 5, We have analyzed hardware synthesis report and found out that all the resource cost is controllable and achievable on Xilinx Virtex6 development board. Then the hardware image segmentation simulation result is compared to image segmentation software result. We showed that the two results are essentially the same, taking into account the random variable limitations. This shows that the EM/MPM hardware design was successfully implemented on chip and had the predicted result. Finally, the speed comparison between the hardware implementation and the software solution is proposed. It is shown that the hardware speeds up the whole 3D image segmentation process by more than 100 times compared to software, and by more than 3 times compared to other hardware segmentation results from the literature.

6.2 Future Work

All the results shown above are either theoretical design analysis and simulation based on Xilinx design platform ISE12.1. Currently we use an image size of 128*128*128 pixels, which is limited by on-chip RAM size. For larger volume 3D image, we can choose the FPGA with more on-chip RAM resource. In future work the design, including the EM algorithm in embedded software, will be implemented on Xilinx hardware. We are now working on including the MPM algorithm as a hardware core which is accessed by the on-chip embedded Microblaze RISC processor. The processor will perform the EM algorithm. The entire system, EM in on-chip embedded software and MPM hardware module, will be tested for accuracy and speed.

LIST OF REFERENCES

LIST OF REFERENCES

[1] L. A.Christopher, E. J.Delp, C. R.Meyer, and P. L.Carson, "3D Bayesian ultrasound breast image segmentation using the EM-MPM algorithm", in *IEEE Trans. Proceedings of the IEEE Symposium on Biomedical Imaging*, 2002.

[2] L. A.Christopher, E. J.Delp, C. R.Meyer, and P. L.Carson "New approaches in 3D Ultrasound segmentation", in *Proceedings SPIE and IST Electronic Ima ging and Technology Conference*,2004.

[3] M. L.Comer and E. J.Delp,"The EM-MPM algorithm for segmentation of textured images: Analysis and further experimental results", in *IEEE Trans. Image Processing*, vol.9, no.10, 2000.

[4] J.C.Li, R.Shekhar and C.Papachristou,"A "brick" caching scheme for 3D medical imaging", *Biomedical Imaging:Nano to Macro*, pp. 563-566,April,15 18, 2004.

[5] T.Schmitt, D.Fimmel,M.Kortke and et al.,"High-speed cone-beam reconstruction an embedded systems approach", *Computer Aided Systems theory-EUROCAST'99*,pp.127-141,Springer,2000.

[6] I.Goddard and M.Trepanier, "High-speed cone-beam reconstruction an embedded systems approach", in *Proceeding SPIE Medical Imaging*,vol.4681,pp.483-491,2002.

[7] S.Coric,M.Leeser,E.Miller,et al., "Parallel-beam backprojection:an FPGA implementation optimized for medical imagine", in *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-progammable gate arrays*, pp.217-226,2002.

[8] B.Cabral, N.Cam, and J.Foran, "Accelerated volume rendering and tomographic reconstruction using texture mapping hardware", in *Proceedings of the 1994 symposium on volume visualization*,pp.91-98,1994.

[9] K.Mueller, "Fast and accurate three-dimensional reconstruction from cone-beam projection data using algebraic methods", *PhD dissertation.The Ohio State University*,1998.

[10] P. V.Dillinger, J.F. Leinen, J. Suslov, S. Patzak, R. Winkler, H. Schwan, "FPGA based real-time image segmentation for medical systems and data processing", *Real Time Conference*, 14th IEEE-NPSS,2005.

[11] K.J.Shanthi, L.R.Ashok, A.S.Anandu, B.Das, "FPGA Implementation of Image Segmentation Processor",*Emerging Trends in Engineering and Technology (ICETET)*, pp.364 - 367, 2009.

[12] S.Malarkhodi, R.S.D.W.Banu, M.Malarvizhi, "VLSI implementation of uterus image segmentation using multi-feature EM algorithm based on Gabor filter: FPGA implementation of uterus image segmentation using multi-feature EM algorithm based on Gabor filter", *Computing Communication and Networking Technologies (ICCCNT)*, 2010.

[13] M.A.Salem, M. Appel, M. Winkler, F. Meffert, "FPGA-based Smart Camera for 3D wavelet-based image segmentation", *Distributed Smart Cameras*, ICDSC, 2008.

[14] Xilinx, *Memory Interface Solution*, User's Guide 086.

[15] Xilinx, "Virtex-6 FPGA Integrated Block for PCI Express (V 1.0)", User's Guide 671,October, 2010.

[16] Rao and Navneet, "Accelerating System Designs Requiring Hign-Bandwidth Connectivity with Targeted Reference Designs", *Xilinx White Paper 359 (v1.0)*,December,2009.