International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-9 Issue-2, December 2019

FPGA Implementation of Mean Shift Algorithm for Real Time Image Segmentation

Anuradha.M.G, Basavaraj L

Abstract: ***Clustering is one of the major steps in image analysis. To speed up the analysis, FPGA implementation of Mean shift algorithm is proposed. The proposed architecture computes the Mean shift operation by using power of two approximations (POTA) for linearization of Gaussian distribution. Also general approach for indexing the window is proposed for selecting the kernel size. The architecture is developed using Verilog HDL and is simulated using Xilinx ISim simulator and synthesized using Virtex 6 FPGA. The Look up table or LUT utilization when the number of input pixels that can be simultaneously processed is varied and hardware cost is analysed. The architecture designed can process four pixels simultaneously with a maximum frequency of 30MHz. The analysis shows that the number of image frames that can be processed vary from 73 to 122 frames per second. This clearly indicates that the developed architecture can be used for various machine learning applications.***

Keywords : Clustering, FPGA, image segmentation, Mean shift algorithm.

I. INTRODUCTION

Clustering is one of the methods in image segmentation to extract the information from an image. In clustering process, similar data samples are grouped together based on the similarity. Due to increase in the data size and iterative nature of the clustering algorithms, the analysis process takes longer time. Hence to speed up the operation of clustering for real time environment, many hardware architectures for various clustering algorithm are proposed in literature.

The related works [1]–[8] presents various architectures for clustering that can be applied for different applications. However, architectures are based on traditional partitional K-means clustering algorithm and fuzzy C-Means clustering algorithm which often results in poor clustering as it depends on the initial conditions. Also iterative assignments and re-assignment process is very much prone to local minimum solution and number of clusters should be specified by the user.

Revised Manuscript Received on December 30, 2019.
***** Correspondence Author
Anuradha.M.G*, Department of Electronics and Communication, *Research Scholar,ATME college of Engineering, Mysore. ** Assistant Professor, JSS Academy of Technical Education Bengaluru. India. Email: anuarun.19@gmail.com
Basavaraj.L, Department of Electronics and Communication, ATME College of Engineering, Mysore.India. Email: basavaraj.atme@gmail.com

The drawback of partitional clustering algorithm is overcome in the Mean shift algorithm which is developed by Fukunaga and Hostetler [9] and was introduced for image analysis by Cheng [10]. The Mean shift algorithm referred as hill climbing algorithm move each data point to its nearest mean depending on the density estimation. Since each data point needs to be converged to its nearest mean, the computational complexity involved in clustering is more than the partitional clustering algorithm.

In recent years, new methods are adopted to reduce the computational complexity in Mean shift algorithm and many architecture for the Mean shift algorithm is developed [10-15]. To speed up the clustering operation, Dang Ba Khac Trio and Tsutomu Maruyama [13] has designed a cache memory to access the windows at arbitrary position and an algorithm for region merging. To reduce the execution time in FPGAs, Stefan Craciun et al. [12] computed Mean shift of the pixels in parallel. To reduce the computations, Amna Tehreem et al. [14] has calculated the distance only for the unique points in a frame.

All the architectures developed till now concentrate on method of accessing and processing the pixels using either Gaussian distribution or flat distribution with look up table values for fixed standard deviation. The memory required to store the Gaussian value will be more. Linear approximation of the Gaussian distribution is proposed in [15]. The multipliers are required to compute the Linear Gaussian distribution which consumes more time and power. In an embedded application, both time and the power dissipation take an important role. Hence, in this paper the linearization of the Gaussian distribution is proposed using Power of two approximations (POTA) where only shifters are used instead of the multiplier. Also to increase the speed of the system, parallel computation is used where up to 256 data elements can be processed together. A general mechanism to index and access the required window is also proposed. To compute the mean of the data points in a window, divider that operate in a single clock cycle is used which reduces the processing time.

The paper is organized as follows. The section 2 briefs the Mean shift algorithm. The architecture designed is described in section 3. Illustration of parallel processing is briefed in section 4. Implementation results are discussed in section 5 and the conclusion is given in section 6.

II. MEAN SHIFT ALGORITHM

Mean shift algorithm is a clustering technique referred as hill climbing algorithm where each data point is moved to its nearest density peak.

Also it does not require any prior information about the initial centres or the number of clusters. The algorithm steps used to build the architecture is briefed in the following section.

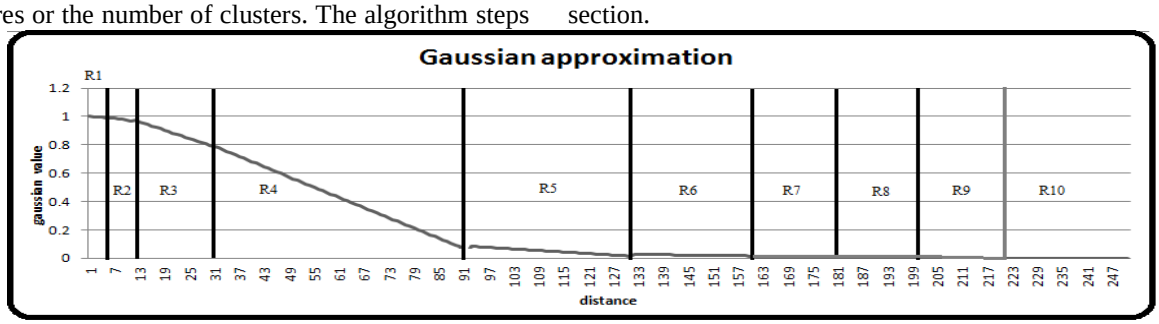

Fig. 1. Proposed Linearization of Gaussian distribution

Table 1: Power of two approximation of Linear Gaussian curve

Regions	Distance (D)	Slope (m)	Intercept (c)	Power Of Two Approximations (POTA)	
				Slope (m)	Intercept (c)
R1	0 – 5	-0.9×10^{-3}	1	$-(2^{-10} - 2^{-14} - 2^{-16})$	2^{0}
R2	6 – 10	-3.9×10^{-3}	1.01	-2^{-8}	$2^{0} + 2^{-6}$
R3	11 – 30	-9.76×10^{-3}	1.08	$-(2^{-9} + 2^{-7})$	$2^{0} + 2^{-4} + 2^{-5} - 2^{-7}$
R4	31 – 89	-0.012	1.15	$-(2^{-7} + 2^{-8})$	$2^{0} + 2^{-3} + 2^{-5}$
R5	90 - 130	-0.0018	0.25	$-(2^{-9} - 2^{-13})$	2^{-2}
R6	131 - 160	-0.0004	0.08	$-(2^{-11} - 2^{-14})$	$2^{-4} + 2^{-5} - 2^{-7}$
R7	161 - 180	-0.000059	0.0115	$-(2^{-14} - 2^{-19})$	$2^{-6} - 2^{-8} - 2^{-12}$
R8	181 - 200	-0.95×10^{-5}	0.0024	$-(2^{-17} + 2^{-19})$	$2^{-9} + 2^{-10} - 2^{-11}$
R9	201 – 220	-0.19×10^{-5}	0.00044	-2^{-19}	$2^{-11} - 2^{-14} + 2^{-16}$
R10	221 - 230	0	0.0000703	0	$2^{-14} + 2^{-16} - 2^{-18} - 2^{-19}$

Consider an image of size N X N, where the data points or pixels in an image is represented as x_1, x_2, and $x_3 \ldots x_{NxN}$. Here each data-point xi has an 8 bit gray scale value. The steps involved in Mean shift algorithm are:

1. Start with the data point x_i.
2. Initialize a window w X w formed by the N neighbouring pixel values of the data point xi.
3. Compute the mean within the window using the Gaussian kernel as represented in equation (1)

$$m(x) = \frac{\sum_{j=1}^{n} xj.e^{\frac{-(xi-xj)^2}{2\sigma^2}}}{\sum_{j=1}^{n} e^{\frac{-(xi-xj)^2}{2\sigma^2}}} \quad (1)$$

4. Move the centre of the window to mean.
5. Repeat steps (3) and (4) till the mean of the window do not change.
6. Repeat steps (1) to (5) for all data points in an image.

Each data pixel in an image is shifted by the amount of gradient of the probability density function.

To compute the mean value for a window, each data point needs to be multiplied by the Gaussian distribution function as seen in equation(1). The exponential value can be computed by using a look up table approach. But look up table require large memory. The Gaussian distribution curve is approximated as straight lines with various slope values and intercepts values as shown in various regions from regions R1 to R9 in Fig 1.

The proposed architecture compute the mean in the window using equation (2) instead of equation (1).

$$m(x) = \frac{\sum_{j=1}^{n} xj.Pj}{\sum_{j=1}^{n} Pj} \quad (2)$$

Here Pj is the linear approximation of Gaussian function given by straight line equation as

$$Pj = Slope * (distance) + intercept \quad (3)$$

Linearization process requires floating point multipliers and adders as seen in equation (3). Hence to speed up the operation, linearization of Gaussian distribution using power of two approximations (POTA) is proposed where the floating point multipliers are replaced by shifters. Based on the distance, 10 regions R1 to R10 proposed for Gaussian distribution shown in Fig 1 has slope and intercept valued as shown in Table 1. For example, if the distance is in between 0 to 5, then in linearization process, distance should be multiplied by $0.9*10^{-3}$ which can be accomplished by shifting the distance by $(2^{-10} - 2^{-14} - 2^{-16})$ which gives the same result of $0.9*10^{-3}$. Hence the floating point number can be multiplied by merely shifting the distance as seen in Table 1 which has accuracy up to three decimal point.

III. PROPOSED ARCHITECTURE

The architecture developed for the Mean shift algorithm is shown in Fig 2. The Mean shift architecture evaluates the probability density function of each of the pixel and moves the pixel in the direction of the gradient. The subsequent sections describe the operation of each block in the architecture used to move the data point to its nearest mode.

A. Input Memory and Window selector

Input memory is used to store the incoming image pixel from the Data-in input. The data pixels are sent row-wise which are stored in the consecutive memory locations.

To process the pixel, the data in a window needs to be selected. Hence the data and its neighbouring pixel address that needs to be processed in a window is calculated using the equation (4).

$$xi = (x + xoff) + (y * yoff) + yindex \quad (4)$$

The processing of the pixel is done by scanning the pixels row-wise. The gradient estimation of each pixel is independent of the other pixels and hence the computation and processing of the pixels can be done in parallel.

For example, if window size is 3X3 and if 4 data pixels at addresses 514,515, 516 and 517 needs to be processed simultaneously, then we need to select all its neighbouring pixels whose address is highlighted in brown, green, pink and orange colours as seen in Figure 3.

If the image size is 256X256, then the neighbours of 514th pixel are the pixels located at address 257,258,259,513,515,769,770 and 771. These pixel locations are selected using equation (4) by spanning x and y values from 0 to 2 as it is a 3X3 window. Any window size can be selected by properly spanning x and y values in equation (4). For example, x and y values can be spanned from 0 to 4 for a window size of 5X5.

The x_{off} value in equation (4) is the first column number of a window that is used to select the proper neighbouring elements. For example, the x_{off} value is set to 1 for window highlighted in brown, x_{off} value is set to 2 for window highlighted in green, and so on.

The y_{off} value in the equation (4) can be selected depending on the image size. It is set to 256 if the image size is 256X256 and set to 512 if image size is 512X512 and so on. It is fixed value depending on the image size.

The y_{index} value is set depending on the image size. Initially, it is set to zero and is incremented after completely processing one single row. The y_{index} value in an image is incremented by 256 if an image size is 256X256 and is incremented by 512 if the image size is 512X512 and so on.

To access the pixel address for the window highlighted in brown, green, purple and orange, the xoff value in equation (3) is 1, 2, 3 and 4 respectively. The yindex value is set to 256 as the elements in the window are starting from second row. The yoff value is set to 256 as it is an example for 256X256 image and x and y values are varied from 0 to 2.

To access the pixel address for the window highlighted in blue, the xoff value in equation (3) is 5. The yindex value is set to 0 as the elements in the window are starting from first row. The yoff value is set to 256 as it is an example for 256X256 image and x and y values are varied from 0 to 2.

The example shown in Figure 3 assumes that an image size is 256X256 and the numbers written are the address location in an input memory.

The data is scanned row-wise from top left pixel to bottom right pixel. The data centring a required pixel is loaded into the local memory. Any element in an input image can be accessed using the equation (4) setting a window around it.

B. Distance Calculator

The distance calculator computes the distance between the centre pixel with all other pixels in a window. Manhattan distance is used to compute the distance as multipliers can be avoided compared to Euclidean distance.

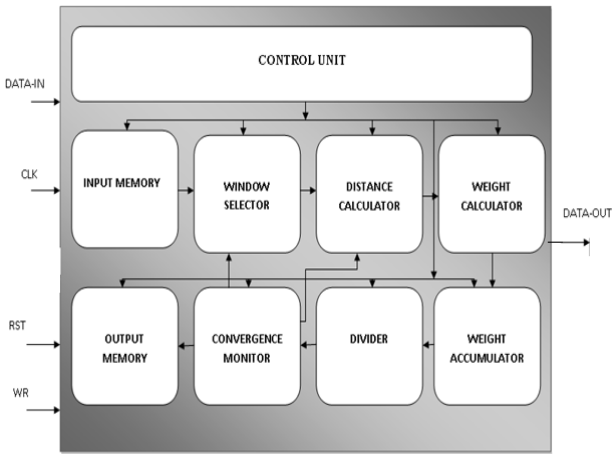

Fig. 2. Proposed Mean shift architecture

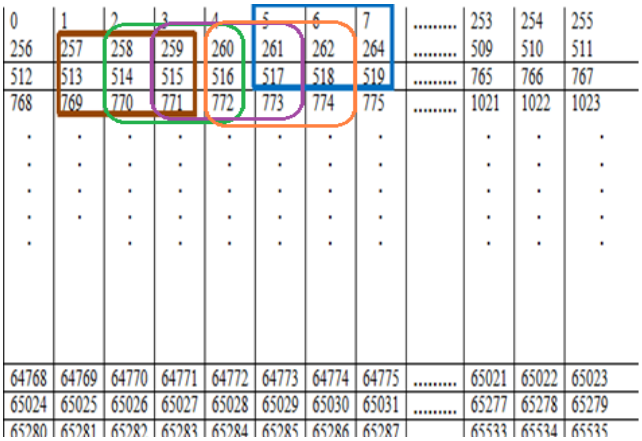

Fig. 3. Illustration of window selection

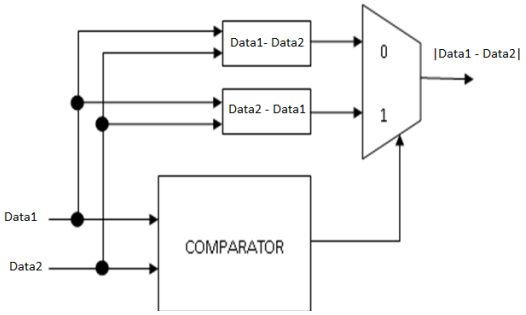

Fig. 4. Distance calculator

The distance calculator used for computing distance is illustrated in Figure 4.

Simultaneously 8 distance calculators are used for processing 1 window of size 3X3. The architecture is tested by varying the number of pixels that can be processed simultaneously. If 4 pixels are simultaneously processed, then we require 32 distance calculators.

C. Weight Calculator

The weight calculator module calculates the weight $p_j * x_j$, the numerator of equation (2) and p_j, the denominator of equation (2) using POTA. Depending on the distance value from the distance calculator module, the distance will be shifted to right using sum of power of two as indicated in Table 1. The registers used to store the result is 28 bits of which upper 8 bits are used to store the integer value and the remaining 20 bits for storing fractional values.

D. Weight Accumulator

The weight calculated by the weight calculator module is stored in the weight accumulator. The weight accumulator consists of two add and accumulator unit for storing the numerator and denominator of equation (2) for each window. For accumulating the denominator and numerator of equation (3), 28 bit accumulators are used. For each window of size 3X3, 8 values are added and accumulated. After adding all the 8 values, if the upper 9^{th} bit in an accumulator is set to 1, then it indicates that the fractional value generated has a value equal to or greater than 0.5 and hence the upper byte which is an integer value is incremented to the next integer value for division process and if the upper 9^{th} bit is zero, then the upper byte value is passed for the division process as it is.

E. Divider

For computing the new mean in a Mean shift process, we need to divide the 8 bit value of $p_j * x_j$ by the 8 bit accumulated weight as seen in (2). The divider designed operates in a single clock cycle obtaining the 8 bit quotient to speed up the clustering operation. The algorithm for obtaining the quotient is given below.

Step1: Set the iteration check value i=0 initially.

Step2: Extend the 8 bit divisor to 16 bit divisor. The new 16 bit divisor is formed by appending one bit zero at MSB and 7 bits zero at LSB of the divisor.

Also extend the 8 bit dividend to 16 bit by concatenating 8 bits of zero's with 8 bit dividend.

Eg: If dividend is 00001001 and divisor is 00000011 then,
New divisor = 0 00000011 0000000 and
New dividend = 00000000 00001001

Step3: Perform Result = New dividend – New divider.

Step 4: Check the result. If the result is negative, then MSB bit of quotient is made zero. If the result is positive, then MSB bit of quotient is made one.

Step 5: Shift the New divider to right by one bit.

Step 6: Increment the iteration check value i.

Step 7: If iteration check value i is less than 8 then go to Step 3 else go to step 8.

Step 8: Stop and return the quotient value.

Since the division operation steps are done using the loop which synthesize to parallel data elements, the division operation is performed using single clock cycle.

F. Convergence monitor

The convergence monitor checks if the quotient obtained from the divider is equal to the previous mean value of a pixel. If same, then the data is converged to its final mean and is stored into the output memory else the new mean is calculated.

IV. ILLUSTRATION OF MEAN SHIFT PROCESS

The mean shift algorithm for processing 4 pixels simultaneously is shown in Fig 5.

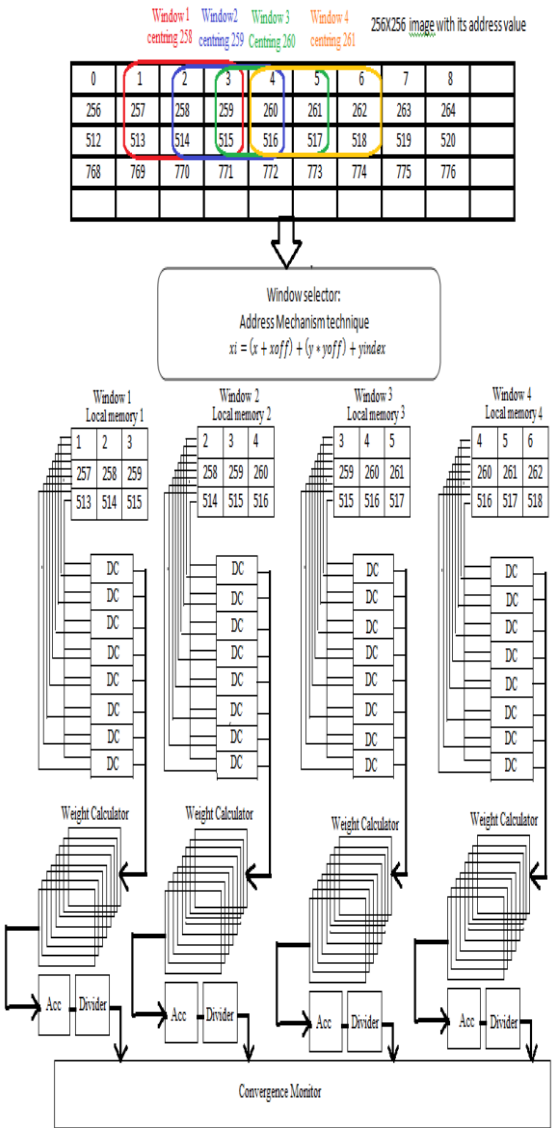

Fig. 5. Illustration of Mean shift operation

The data from the input memory passes from one module to another module in sequentially but the computations within the modules window selector, distance calculator, weight calculator, weight accumulator and divider are computed in parallel such that they compute the said operation in a single clock cycle.

In the first clock cycle, the window selector selects 4 pixels that need to be processed and its 8 neighbouring pixels for a 3X3 window and stores it in four local memories as shown in Fig 5.

In the second clock cycle, the distance between the centre pixel and its neighbours are computed in parallel using 8 distance calculators. Since 4 pixels are processed simultaneously using 4 local memories, there are 32 distance calculators which compute the absolute distance between the centre pixel and all its neighbouring pixels in parallel.

In the third clock cycle, the distance computed is given to the weight calculator to compute the Gaussian distribution f the pixel. 8 weight calculators are required for each window and hence 32 weight calculators operate in parallel.

The numerator and denominator of equation (2) are stored in the weight accumulator unit. Since 4 pixels are processed at a time, there are 4 weight accumulator units operating in parallel. The divider outputs new mean by dividing the values from weight accumulator unit which are the numerator and denominator of equation (2).

Each module in the architecture outputs the data in a single clock cycle and hence five clock cycles are required to compute new mean.

The convergence of each window is independent and hence after the mean is converged in each window, the window is shifted by incrementing x_{off} by four.

V. RESULTS AND DISCUSSIONS

The proposed Mean shift architecture using POTA is built using Verilog HDL and tested using I-Sim simulator and MATLAB. The proposed Mean shift algorithm is implemented using Xilinx Virtex 6 FPGA.

The first part of verification is the simulation of the proposed architecture for the mean shifting. The intensity values from an image are extracted using MATLAB and the extracted values are fed for the Verilog Module for testing the hardware architecture. The simulation result for the window slice with step by step illustration of Mean shift algorithm is as shown in Fig 6. A part of an image input stored in an input memory is shown in Fig 6a. Each input pixel need to be converged to its nearest density point for which the neighbouring pixels need to be accessed. The 3X3 window centring the data pixel 113 is stored in a local memory for processing is shown in Fig 6b. The distance computed between the centre pixel and its neighbouring is illustrated in Fig 6c which matches the theoretical value as shown. Fig 6d and Fig 6e shows the weight calculation result of the window using the POTA method. The result shows that the data obtained matches the theoretical result.

Fig. 6a. Input memory

	0	1	2	3	4	5	6	7	8	9
0	115	116	106	107	108	111	104	109	110	111
256	109	113	109	106	105	112	110	108	108	97
512	106	109	111	109	107	113	104	107	110	100
768	103	113	102	109	113	112	113	100	110	104
1024	105	113	107	113	109	118	110	109	109	101
1280	111	114	107	106	112	113	107	102	99	97
1536	112	107	106	105	103	103	99	105	108	108
1792	107	107	104	113	107	111	107	114	114	126
2048	115	126	135	128	126	128	132	146	140	152
2304	150	152	152	155	157	162	163	165	169	170

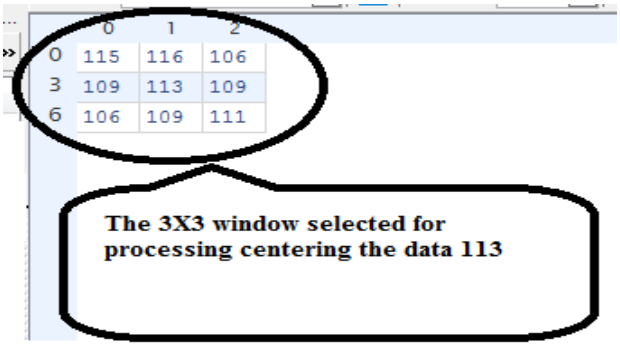

Fig. 6b. 3X3 memory

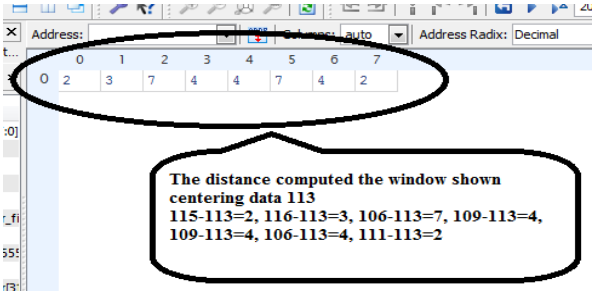

Fig. 6c. Distance calculator

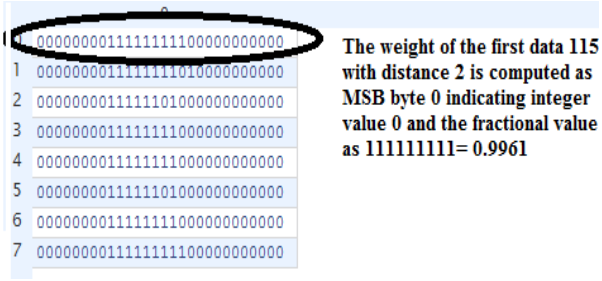

Fig. 6d. Weight calculator

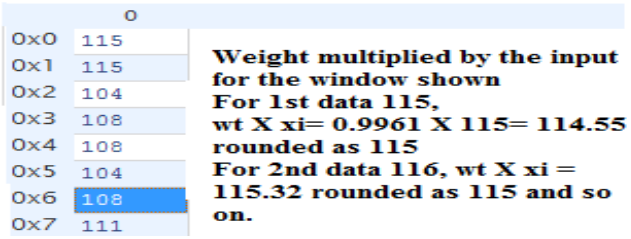

Fig. 6e. Weight multiplier performing Weight X input using POTA

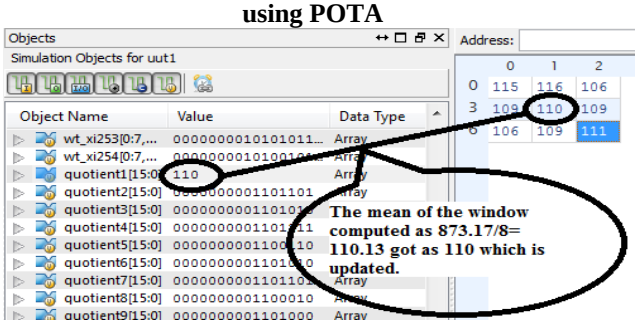

Fig. 6f. Re-computation of mean in a window

Fig 6f shows the re-computation of mean in the window which is again updated from the divider module to the local memory for further processing.

The numerical result obtained from the architecture developed using Verilog HDL is visualized using the MATLAB tool. The visual results of Mean shift clustering result is shown in Fig 7. To test the architecture, standard 256X256 images were taken. The architecture was tested using Lena image, pepper image and camera man image as it covers the mixture of flat regions, shading and texture in an image. The visual segmentation results obtained with and without using the sum of power of two approximation when visualized were alike.

The hardware cost in the FPGA is analysed by varying the input that can be simultaneously varied from 4 to 256.

The LUTs used in the design when the number of pixels that can be simultaneously processed is varied from 4 to 6 is shown in Table 2. The Virtex 6 has 124800 Slice LUTs. When the input that can be processed is 4, 32% of the resource is utilized. When the input that can be processed is 5, 45% of the resource is utilized and when the input that can be processed is 5, 57% of the resource is utilized. The resource utilized is increasing linearly with the input that can be processed in parallel. If the number of pixels that can be processed is made more than ten, then the LUT slice register required is more than the available FPGA resource.

The maximum speed of the architecture using linearization of Gaussian distribution without using the power of two approximation and using power of two approximation was carried out and through the result it was observed that the architecture using POTA operate faster compared to architecture without using the shifter as seen in Fig 7. Also the maximum frequency that the architecture operates varies from 30MHz to 21 MHz when processing 4 pixels simultaneously to 16 pixels simultaneously using POTA. As seen, the POTA architecture works faster than just linearization of Gaussian approximation without POTA.

The resource utilized in FPGA with and without POTA is shown in Fig 8. The number of register pairs and the logic used increases with increase in the number of pixel used for parallel processing. The number of resource utilized is more in the architecture that use POTA compared to the architecture without using the sum of two approximation due to large number of shifters required in parallel processing as seen in Fig 8.

Fig. 7. Visual results of Mean Shift Clustering

Table 2: LUT Utilization summary for varied number of inputs that can be simultaneously processed

	N=4	N=5	N=6
Slice Register	2434	3005	3544
Slice LUT	40482	56564	72259
LUT flops USED	40192	57105	72939
DSP blocks	640	640	640

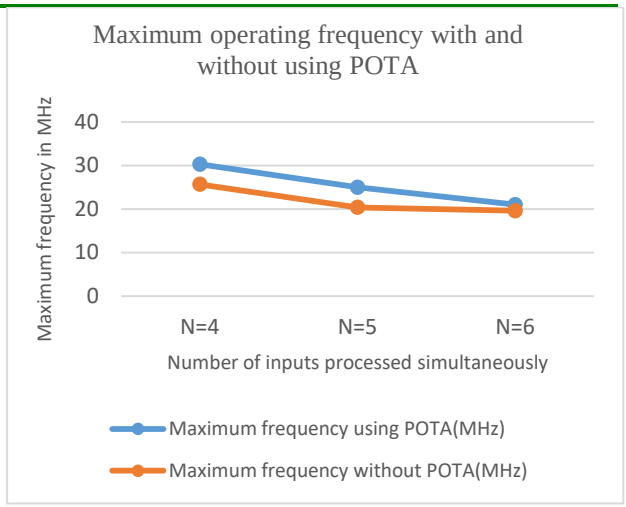

Fig. 7. Maximum operating speed of architecture with and without Power of two approximation

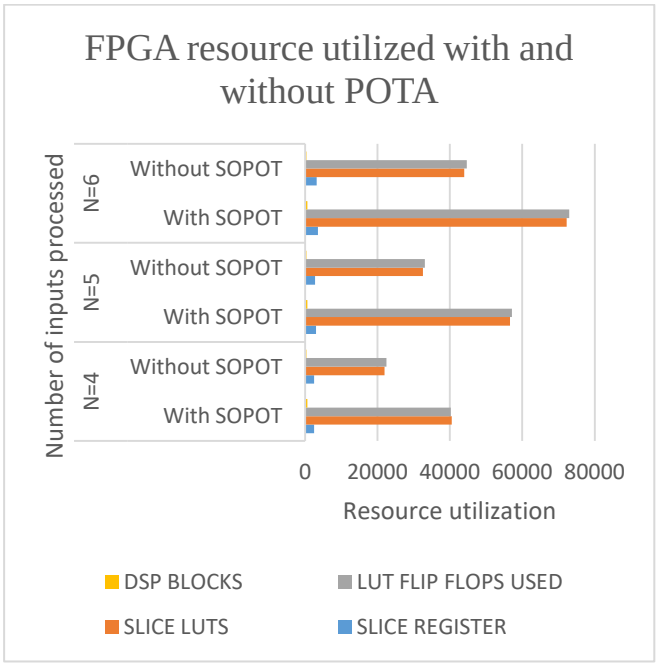

Fig. 8. FPGA resource utilization with and without Power of two approximation

The time constraint in real time environment for the proposed architecture is analyzed in the section below.

Each hardware module in the proposed architecture computes the said operation in a single clock cycle. If four data's are processed simultaneously, then all four data needs to go through distance calculator unit, weight calculator unit,

weight accumulator, and divider and convergence monitor units for computing the window shift and checking for convergence. Since each module take 1 clock cycle to compute, all the four data take 5 clock cycle to complete the iteration. If mean has converged, then it will select the new window, else iterate once again through the distance calculator, weight calculator, weight accumulator, divider and convergence monitor. In an average 3 to 5 iterations are required to converge to final mean. Hence for the mean to converge, it takes (5 clock cycles X 3 to 5 iterations) 15 to 25 clock cycles. Since each pixel should be converged to its mean in a window, we require iterating the computing blocks for 16384 times if 4 data are processed simultaneously for an image size of 256X256. The total clock cycles required to complete one image frame varies from 245760 to 409600 clock cycles. Since minimum period is 33.33ns, we can compute the mean shift operation of an image within 8.192ms to 13.65ms. This indicate that the number of image frames that can be processed vary from 73 to 122fps.

Also for a window size of 3X3, it is observed that the number of iteration required to converge takes less than or equal to 3 clock cycles. Hence on an average 130 image frames can be processed per second. This indicates that the proposed architecture can be used for real time image processing.

VI. CONCLUSION

FPGA implementation of Mean shift algorithm is proposed in the paper. The architecture developed proposes linear approximation of the Gaussian kernel using the power of two approximations to reduce the floating point multipliers and the look up table memory overhead.

Also the architecture uses single clock cycle divider to compute the new mean. A new method to access NXN window is proposed for performing the mean shift operation. The architecture developed is simulated using Xilinx I-Sim simulator and implemented using Virtex 6 FPGA. The proposed architecture can work at maximum frequency of 30MHz using the FPGA platform and due to parallel architecture and shifters usage, the proposed architecture can process up to 122 image frames per second and hence is suited for real time applications.

The architecture is tested for 3X3 window size. But a generalized method is proposed to access NXN window. As a future development, area and power analysis for varying window size needs to be carried out.

REFERENCES

1. Saegusa T, Maruyama T.: ‘An FPGA implementation of real-time K-means clustering for color images’. Journal of Real-Time Image Processing. 2007 Dec 1;2(4):309-18.
2. S Filho, A. Gda, Alejandro César Frery, Cristiano C. de Araujo, Haglay Alice, Jorge Cerqueira, Juliana A. Loureiro, Manoel Eusebio de Lima, Mdas GS Oliveira, and Michelle Matos Horta.: "Hyperspectral images clustering on reconfigurable hardware using the k-means algorithm." In *16th Symposium on Integrated Circuits and Systems Design, 2003. SBCCI 2003. Proceedings.* pp. 99-104. IEEE, 2003.
3. Maruyama, Tsutomu.: "Real-time k-means clustering for color images on reconfigurable hardware." In *18th International Conference on Pattern Recognition (ICPR'06)*, vol. 2, pp. 816-819. IEEE, 2006.
4. Chen, Tse-Wei, Chih-Hao Sun, Jiun-Ying Bai, Han-Ru Chen, and Shao-Yi Chien.: "Architectural analyses of K-Means silicon intellectual property for image segmentation." In *2008 IEEE International Symposium on Circuits and Systems*, pp. 2578-2581. IEEE, 2008.
5. Chen, Tse-Wei, and Shao-Yi Chien.: "Bandwidth adaptive hardware architecture of K-means clustering for video analysis." *IEEE transactions on very large scale integration (VLSI) systems* 18, no. 6 (2009): 957-966.
6. Chen, Tse-Wei, and Shao-Yi Chien. "Flexible hardware architecture of hierarchical K-means clustering for large cluster number." *IEEE transactions on very large scale integration (VLSI) systems* 19, no. 8 (2010): 1336-1345.
7. Li, Hui-Ya, Wen-Jyi Hwang, and Chia-Yen Chang. "Efficient fuzzy C-means architecture for image segmentation." *Sensors*11, no. 7 (2011): 6697-6718.
8. Anuradha MG, Basavaraj L.: ‘Design and Implementation of High Speed VLSI Architecture of Online Clustering Algorithm for Image Analysis’. In Data Engineering and Intelligent Computing 2018 (pp. 197-206). Springer, Singapore.
9. Fukunaga, Keinosuke, and Larry Hostetler. "The estimation of the gradient of a density function, with applications in pattern recognition." *IEEE Transactions on information theory* 21, no. 1 (1975): 32-40.
10. Parzen, Emanuel. "On estimation of a probability density function and mode." *The annals of mathematical statistics* 33, no. 3 (1962): 1065-1076.
11. Cheng, Yizong. "Mean shift, mode seeking, and clustering." *IEEE transactions on pattern analysis and machine intelligence* 17, no. 8 (1995): 790-799.
12. Craciun, Stefan, Robert Kirchgessner, Alan D. George, Herman Lam, and Jose C. Principe. "A real-time, power-efficient architecture for mean-shift image segmentation." *Journal of Real-Time Image Processing* 14, no. 2 (2018): 379-394.
13. Trieu, Dang Ba Khac, and Tsutomu Maruyama. "Real-time color image segmentation based on mean shift algorithm using an FPGA." *Journal of Real-Time Image Processing* 10, no. 2 (2015): 345-356.
14. Tehreem, Amna, Sajid Gul Khawaja, Asad Mansoor Khan, Muhammad Usman Akram, and Shoab A. Khan. "Multiprocessor architecture for real-time applications using mean shift clustering." *Journal of Real-Time Image Processing* (2017): 1-14.
15. Tsai, Chang-Hung, Hui-Hsuan Lee, Wan-Ju Yu, and Chen-Yi Lee. "A 2 GOPS quad-mean shift processor with early termination for machine learning applications." In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 157-160. IEEE, 2014.

AUTHORS PROFILE

Anuradha.M.G received B.E degree from Visvesvaraya Technological university in Electronics and Communication in the year 2002 and M.Tech degree in VLSI and Embedded system design from Visvesvaraya Technological University in the year 2008. Currently she is a research student in ATME college of Engineering and working as Assistant professor in JSS Academy of Technical Education, Bengaluru. She has published more than 15 research articles in various International Conferences and journals. Her research intrest is on pattern recognition, Image clustering and its related VLSI architectures.

Dr. L. Basavaraj L received B.E degree from university of Mysore in Electrical and Electronics in the year 1989, M.E degree from University of Dharwad in Digital Electronics in the year 1996 and Ph.D Degree in Electronics from University of Mysore. He has 31 years if teaching experience and has published more than 40 papers in various International conferences and journals. He has received best teacher award from ISTE. He is currently working as professor and principal in ATME college of Engineering, Mysore. His research interest includes image processing and hand writing analysis.