Volume : 3 | Issue : 5 | May 2014 • ISSN No 2277 - 8179

Research Paper

Design and Implementation of GABOR Type Filter on FPGA

Engineering

KEYWORDS : gabor filter, floating point, field programmable gate arrays(FPGAs)

Sarika R | PG Scholar, Vedavyasa Institute of Technology, Malappuram, Kerala, India

ABSTRACT

The gabor filter architecture used in this paper is suitable for real time applications with high pixel rates. The structure is capable of processing images using floating point computation. Xilinx ISE suite is used for simulation. Verilog HDL is used for mapping the algorithm in VLSI and FPGA implementation is done on SPARTAN- 3E.

Introduction

In image processing Gabor Filters are linear filters mainly used for edge detection. The frequency and orientation selectivity property of these filters are similar to the human visual system. In this brief we are using the floating point computation technique for the design of gabor filters. Floating point computation is more advantageous than any other computation technique, because of its accuracy.

In this paper we will discuss about gabor filters, floating point conversion of numbers, multiplication and addition of floating point numbers used in the basic gabor filter architecture. Finally the simulation is done using Xilinx and FPGA implementation using SPARTAN 3E.

Gabor filters

Gabor filters are used for edge detection because of its frequency and orientation selectivity properties. In spatial domain, a gabor filter is a Gaussian kernel function modulated by a sinusoidal plane wave. The simple cells in the visual cortex system of humans can be modulated using gabor functions. So image analysis of gabor filters are similar to perception of human eyes.

Definition of Gabor Filter

The impulse response of gabor filter can be represented by complex valued convolution kernels.

$$g(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{-x^2+y^2}{2\sigma^2}} e^{j(\omega_x x + \omega_y y)}$$

$\boldsymbol{\sigma}$, $\boldsymbol{\omega}_x$ and $\boldsymbol{\omega}_y$ are real constants. The even kernel in the function is cosine modulated and odd kernel is sine modulated. The filter will have a real and imaginary components.

The inputs, state and output of an image is u_{ij}, x_{ij} and y_{ij} respectively. In our architecture pipelined datapaths are used. If the input image pixel values and the initial states satisfy the equation

$|u_{ij}|<=1$, $|x_{ij}^R(0)|<=1$, $|x_{ij}^I(0)|<=1$ for i= 1.....M, j=1.....N.

Then the real and imaginary parts of the output image pixel values are kept in the same range.

From [6] and [7] we get the values of α_x, α_y, β_x and β_y.

The values of x^R and x^I are in between -1 and +1. The values of

$$\alpha_x = \frac{\cos\omega x u}{4+\lambda^2}, \quad \beta_x = \frac{\cos\omega y u}{4+\lambda^2}, \quad \alpha_y = \frac{\sin\omega x u}{4+\lambda^2}, \quad \beta_y = \frac{\sin\omega y u}{4+\lambda^2}$$

$$b = \frac{\lambda^2}{4+\lambda^2}$$

α_x, α_y, β_x , β_y and b are in between -0.25 and +0.25. these filter coefficients are changed according to the values of $\boldsymbol{\omega}_x$, $\boldsymbol{\omega}_y$ and $\boldsymbol{\lambda}$.

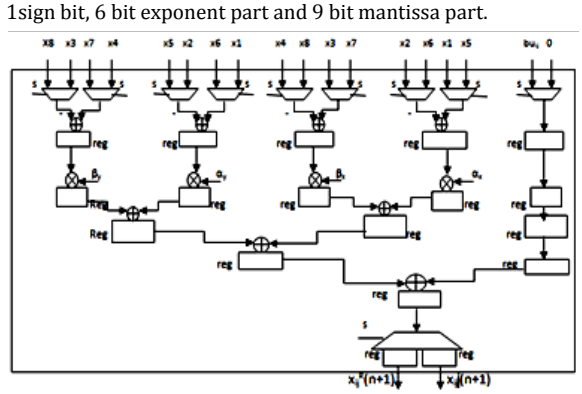
Proposed Method

The fig1 shows the logical block diagram of a GTF(Gabor Type Filter) unit. Inputs for the given system are x1,x2,....x8.Another input is b, which is used for the determination of real part only.

For the multipliers the inputs are βx, βy, αx, αy. From the reference papers we get ωx0 = 30 degree, ωy0 = 60 degree, λ = 0.5. By using these values we get βx = 0.1176, βy =0.2037 , αx = 0.2037, αy= 0.1176 and b= 0.05882. In this architecture we are using floating point addition and multiplication. Floating point computation is more precise and accurate than any other arithmetic computation. X is the state and it ranges from -1<x<1.

Floating point conversion

Consider a 16 bit number. The 16 bit number is divided into 1sign bit, 6 bit exponent part and 9 bit mantissa part.



1bit 6bit 9 bit

Fig1. Logical block diagram of gabor filter

1 100000 100000000

First value x1= -1, the sign bit is 1, because of negative value. The exponent part of -1 is obtained by the below procedure

-1 → 1.00000

One right shift

1.0 → 0.100000

Base value is $2^{6-1}-1 = 2^5-1 = 32-1= 31$.The exponent part is Number of shifting + Base value. Here the number of shift is 1. So the exponent is 1 + 31= 32 → 100000 6 bit exponent.

Next is the mantissa part which is the 9 bit value 100000000. Then the 16 bit floating point value for -1 is

Next value x2= 0.5.The binary value of 0.5 is 0.1.the sign bit is 0, because the number is positive. The exponent part is number of shift + base value. Here there is no shifting, so the exponent is the base value itself, ie $2^{6-1}-1 = 31$ → 11111, for 6 bit exponent 011111. The mantissa part is 100000000. Then the 16 bit floating point value of 0.5 is

0 011111 100000000

Next value is 0.25. Its binary value is 0.01. So one left shift is needed. Here the sign bit is 0, since 0.25 is a positive number. Because of left shifting the exponent value is (base value – 1), ie 31- 1 = 30, binary value is 11110. The exponent 6 bit value is 011110. The mantissa part is 100000000

0 011110 100000000

Next value is x4= 1. Here the sign bit is 1, because 1 is a positive value. The exponent and mantissa is same as that of x1.

0 100000 100000000

Next we have to convert the values of βx = 0.1176, βy =0.2037 , αx = 0.2037, αy= 0.1176 and b= 0.05882 to floating point numbers. First is the value of b, the binary value of b = 0.0588235 → 0.0000111100001111.

For converting b to floating point value 4 left shifting is needed. The sign bit is 0. The exponent value is (base value – 4)= 31-4 = 27. The binary value of 27 is 11011.the

mantissa part is 111100001.

0 011011 111100001

From x1 to x8

State	Sign bit	Exponent	Mantissa
X1= -1	1	100000	100000000
X2 = 0.5	0	011111	100000000
X3= 0.25	0	011110	100000000
X4= 1	0	100000	100000000
X5= -1	1	100000	100000000
X6= 0.5	0	011111	100000000
X7= 0.25	0	011110	100000000
X8= 1	0	100000	100000000

Table1

Next value is αx = 0.2037. the binary value of 0.2037 is 0.001101010110100111. For converting this value to floating point, 2 left shifting is needed. Sign bit is 0, the exponent value is (base value – 2)= 31-2 = 29 → 11101 . The mantissa part is 110101011

0 011101 110101011

Since βy = αx = 0.2037, the floating point value of βy is same as αx,0 011101 110101011.

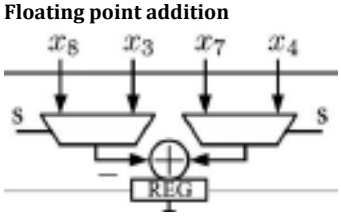
Next is the value of βx = 0.1176. the binary value is 0.0001111000011110. For converting this to floating point value 3 left shifting is needed. Sign bit is 0.the exponent value is (base value – 3) = 31-3 = 28 → 11100. The mantissa part is 111100001

0 011100 111100001

The value of βx = αy= 0.1176, so the floating point value of αy is same as that of βx.

1 100000 011000000

Floating point addition



When the select line s= 0, x8 and x7 is selected by the multiplexer. First x8 is multiplied by -1. So the value of x8 becomes 1 100000 100000000. The value of x7 is 0 011110 100000000.

x8 1 100000 100000000

x7 0 011110 100000000

x8 → Y1 , x7 → Y2

Y3= Y1 + Y2

For addition of two floating point numbers make the exponent part same. The exponent part of x8, exp1 is 32 and the exponent part of x7, exp2 is 30.the difference of exp1 and exp2 (exp1 – exp2) is 32-30 = 2. Add 2 to exp 2, so that exp1 = exp2.since 2 is added to exponent part of x7, the mantissa part of x7 is right shifted by 2 units, 100000000 becomes 001000000. Since the sign of two numbers are different, we have to subtract mantissa2 from mantissa1 (mantissa1-mantissa2). Sign of Y1 + Y2 is the sign of highest exponent, 1

Mantissa1- mantissa2 → 100000000 – 001000000 = 011000000

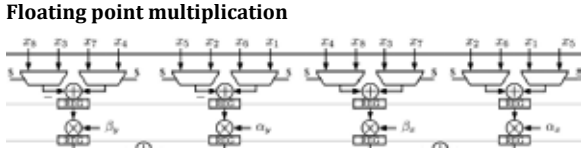
Y3= Y1 + Y2 is

Like this way when s= 0 , Y1 x3, Y2 x4, then Y3 is Y1+ Y2 ie x3 + x4.

1 100010 010100000

Addition operations for each multiplexers is carried out in this way and the values are stored in the registers. Then we have to do the multiplication operation of floating point numbers.

Floating point multiplication



Next we have to do the floating point multiplication. Consider the multiplier in the architecture. The inputs to the multiplier is Y3 → 1 100000 100000000 and

βy → 0 011101 110101101. The multiplier output is Y4 = Y3 * βy.

The sign bit of the product is obtained by XOR operation. The sign bit of Y4 is, sign bit of Y3 (XOR) sign bit of βy, ie 1 ^ 0 = 1. The sign bit of Y4 is 1.

Next is the exponent part.

Exp1- base value1 = 32- 31 =1

Exp 2- base value2= 32-29= 2

Therefore the final exponent is 1+2+31= 34, 100010

The mantissa part is obtained by normal multiplication of two binary numbers. The output of the multiplier, Y4 is

Results

The proposed structure for gabor filter is designed using floating point computation. The inputs given to the filter is x1 to x8. The given input ranges from -1 to +1. These values are converted to floating point values using the verilog program. Then the addition operator adds the values and multiplier gives the product of the values given to it. Finally we will get the real and imaginary part of the image given to the gabor filter.The simulation is done using Xilinx for getting real and imaginary part. Finally the FPGA implementation is obtained on SPARTAN 3E.

Conclusion and Future work

In this brief, the GTF structure is proposed using floating point method. By introducing floating point computation we will get an accurate output for the given input. We can use real number

values instead of integers used in fixed point arithmetic. These floating point computations are done using the verilog codes. Finally the implementation of the GTF on FPGA is obtained.

Our future work includes finding the design details of the gabor filter using floating point computation and to compare these values with the fixed point operation.

Acknowledgement

The author gratefully acknowledges the support and facilities provided by Department of ECE, Vedavyasa Institute of Technology under University of Calicut. Author also extends her acknowledgement to Ms Anupama Varghese T(Asst. Professor, Dept of ECE) for her immense help during the course of project.

REFERENCE

[1] X. Wang and B. Shi, "GPU implementation of fast Gabor filters," in Proc. IEEE ISCAS, May 30–Jun. 2, 2010, pp. 373–376. | [2] E. Norouznezhad, A. Bigdeli, A. Postula, and B. Lovell, "Robust object tracking using local oriented energy features and its hardware/software implementation," in Proc. 11th ICARCV, Dec. 2010, pp. 2060–2066. | [3] Y. Cho, S. Bae, Y. Jin, K. Irick, and V. Narayanan, "Exploring Gabor filter implementations for visual cortex modeling on FPGA," in Proc. Int. Conf. FPL, Sep. 2011, pp. 311–316. | [4] J. Liu, S.Wang, Y. Li, J. Han, and X. Zeng, "Configurable pipelined Gabor filter implementation for fingerprint image enhancement," in Proc. 10th IEEE ICSICT, Nov. 2010, pp. 584–586. | [5] O. Cheung, P. Leong, E. Tsang, and B. Shi, "A scalable FPGA implementation of cellular neural networks for Gabor-type filtering," in Proc. IJCNN, 2006, pp. 15–20. | [6] E. Saatci, E. Cesur, V. Tavsanoglu, and I. Kale, "An FPGA implementation of 2-D CNN Gabor-type filter," in Proc. 18th ECCTD, Aug. 2007, pp. 280–283. | [7] E. Cesur, N. Yildiz, and V. Tavsanoglu, "An improved FPGA implementation of CNN Gabor-type filters," in Proc. IEEE ISCAS, May 2011, pp. 881–884. | [8] N. Yildiz, E. Cesur, and V. Tavsanoglu, "A new control structure for the pipelined CNN processor arrays," in Proc. 12th Int. Workshop CNNA, Feb. 2010, pp. 1–4. | [9] E. Cesur, N. Yildiz, and V. Tavsanoglu, "Architecture of the next generation real time CNN processor: RTCNNP-v2," in Proc. Int. Symp. NOLTA, Krakow, Poland, Sep. 2010. | [10] L. Chua and T. Roska, Cellular Neural Networks and Visual Computing: Foundation and Applications. Cambridge, U.K.: Cambridge Univ. Press, 2002. | [11] B. Shi, "Gabor-type filtering in space and time with cellular neural networks," IEEE Trans. Circuits Syst. I, Fundam. Theory Appl., vol. 45, no. 2, pp. 121–132, Feb. 1998. | [12] E. Saatci and V. Tavsanoglu, "On the optimal choice of integration timestep for raser simulation of a CNN for gray level image processing," in Proc. IEEE Int. Symp. ISCAS, 2002, vol. 1, pp. I-625–I-628. | [13] V. Tavsanoglu, "Jacobi's iterative method for solving linear equations and the simulation of linear CNN," in Proc. 10th Int. Workshop CNNA, Aug. 2006, pp. 1–5. | [14] D. Dunn and W. Higgins, "Optimal Gabor filters for texture segmentation," IEEE Trans. Image Process., vol. 4, no. 7, pp. 947–964, Jul. 1995.