An FPGA Implementation Of 2-D CNN Gabor-Type Filter

Ertugrul Saatci*, Evren Cesur**, Vedat Tavsanoglu**, Izzet Kale***
*Department of Computer Engineering, Istanbul Kultur University, Istanbul, Turkey
Email: e.saatci@iku.edu.tr
**Department of Electronics and Communications Engineering, Yildiz Technical University, Istanbul, Turkey
Email: {ecesur, tavsanav}@yildiz.edu.tr
*** Department of Electronic Systems, University of Westminster, London, UK

***Abstract*—A field programmable gate array (FPGA) implementation of the Gabor-type filter is presented. The implementation uses the forward Euler approximation with optimal step size to solve the CNN cell-state equation. The FPGA is implemented on Xilinx Spartan XC3S400 device using 219 slices. An image of dimension 60 x 60 can be processed without using any external RAM only with the block RAM.**

I. INTRODUCTION

Gabor-type filter [1] has recently been implemented on a field programmable gate array (FPGA) [2] which is based on a structure made up of two layers where one layer is assigned to the even part and the other layer is to the odd part of the impulse response.

In this paper the CNN cell-state equation is solved using the forward Euler approximation with the optimal step size given in [3]. An alternative arrangement of the resulting discrete-time equation enables the development of a single layer structure.

II. THE CNN CELL STATE EQUATION

The state equation for the ij-th cell of a linear CNN is given as [4]:

$$\dot{x}_{ij} = A \otimes X_{ij} + B \otimes U_{ij} \qquad (1)$$

where x_{ij} and u_{ij} are the state and the input of the ij-th cell, respectively. The template dot-product is defined as :

$$A \otimes X_{ij} = \begin{bmatrix} a_{-1,-1} & a_{-1,0} & a_{-1,1} \\ a_{0,-1} & a_{0,0} & a_{0,1} \\ a_{1,-1} & a_{1,0} & a_{1,1} \end{bmatrix} \otimes \begin{bmatrix} x_{i-1,j-1} & x_{i-1,j} & x_{i-1,j+1} \\ x_{i,j-1} & x_{i,j} & x_{i,j+1} \\ x_{i+1,j-1} & x_{i+1,j} & x_{i+1,j+1} \end{bmatrix}$$

$$B \otimes U_{ij} = \begin{bmatrix} b_{-1,-1} & b_{-1,0} & b_{-1,1} \\ b_{0,-1} & b_{0,0} & b_{0,1} \\ b_{1,-1} & b_{1,0} & b_{1,1} \end{bmatrix} \otimes \begin{bmatrix} u_{i-1,j-1} & u_{i-1,j} & u_{i-1,j+1} \\ u_{i1,j-1} & u_{i,j} & u_{i,j+1} \\ u_{i+1,j-1} & u_{i+1,j} & u_{i+1,j+1} \end{bmatrix}$$

where A and B are the feed-back and feed-forward input templates, respectively.

Assuming that A and B templates are complex-valued and, the input image U is real-valued, we can write (1) in the following form:

$$\dot{x}_{ij}^R + j\dot{x}_{ij}^I = A_R \otimes X_{ij}^R - A_I \otimes X_{ij}^I + j\left(A_R \otimes X_{ij}^I + A_I \otimes X_{ij}^R\right)$$

$$\begin{bmatrix} \dot{x}_{ij}^R \\ \dot{x}_{ij}^I \end{bmatrix} = \begin{bmatrix} A_R & -A_I \\ A_I & A_R \end{bmatrix} \otimes \begin{bmatrix} X^R \\ X^I \end{bmatrix} + \begin{bmatrix} B_R \\ B_I \end{bmatrix} \otimes \begin{bmatrix} U^R \\ U^I \end{bmatrix} \qquad (2)$$

The feed-back and feed-forward templates for the Gabor-type filters are given as:

$$A = \begin{bmatrix} 0 & e^{-j\omega_{y0}} & 0 \\ e^{j\omega_{x0}} & -\left(4+\lambda^2\right) & e^{-j\omega_{x0}} \\ 0 & e^{j\omega_{y0}} & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \lambda^2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

which yield

$$A_{\mathrm{R}} = \begin{bmatrix} 0 & \cos\omega_{y0} & 0 \\ \cos\omega_{x0} & -(4+\lambda^2) & \cos\omega_{x0} \\ 0 & \cos\omega_{y0} & 0 \end{bmatrix}, \tilde{A}_{\mathrm{R}} = \begin{bmatrix} 0 & \cos\omega_{y0} & 0 \\ \cos\omega_{x0} & 0 & \cos\omega_{x0} \\ 0 & \cos\omega_{y0} & 0 \end{bmatrix}$$

$$\tilde{A}_{\mathrm{I}} = \begin{bmatrix} 0 & \sin\omega_{y0} & 0 \\ -\sin\omega_{x0} & 0 & \sin\omega_{x0} \\ 0 & -\sin\omega_{y0} & 0 \end{bmatrix}$$

where (~) stands for surround template [4].

This research has been supported by Yildiz Technical University Scientific Research Projects Coordination Department. Project Number: 25-04-03-01

The application of forward Euler integration algorithm to (2) and using the optimal numerical integration step-size given in [3], yields:

$$\begin{bmatrix} x_{ij}^R(k+1) \\ x_{ij}^I(k+1) \end{bmatrix} = \frac{1}{4+\lambda^2}\left\{ \begin{bmatrix} \tilde{A}_R & \tilde{A}_I \\ -\tilde{A}_I & \tilde{A}_R \end{bmatrix} \otimes \begin{bmatrix} X_{ij}^R(k) \\ X_{ij}^I(k) \end{bmatrix} + \begin{bmatrix} B_R \\ B_I \end{bmatrix} \otimes \begin{bmatrix} U^R \\ U^I \end{bmatrix} \right\} \quad (3)$$

where

$$B_R = B,\ B_I = 0\ ,\ U_{ij}^R = U_{ij} \text{ and } U_{ij}^I = 0.$$

Now defining the vectors:

$$\boldsymbol{x}_{i-1,j}(k) = \begin{bmatrix} x_{i-1,j}^R(k) \\ x_{i-1,j}^I(k) \end{bmatrix}, \boldsymbol{x}_{i,j+1}(k) = \begin{bmatrix} x_{i,j+1}^R(k) \\ x_{i,j+1}^I(k) \end{bmatrix},$$

$$\boldsymbol{x}_{i+1,j}(k) = \begin{bmatrix} x_{i+1,j}^R(k) \\ x_{i+1,j}^I(k) \end{bmatrix}, \boldsymbol{x}_{i,j-1}(k) = \begin{bmatrix} x_{i,j-1}^R(k) \\ x_{i,j-1}^I(k) \end{bmatrix}$$

(3) can also be written in the form:

$$\begin{bmatrix} x_{ij}^R(k+1) \\ x_{ij}^I(k+1) \end{bmatrix} = \frac{1}{4+\lambda^2}\left\{ \begin{array}{l} \begin{bmatrix} \cos\omega_{x0} & -\sin\omega_{x0} \\ \sin\omega_{x0} & \cos\omega_{x0} \end{bmatrix} \begin{bmatrix} x_{i-1,j}^R(k) \\ x_{i-1,j}^I(k) \end{bmatrix} + \\ \begin{bmatrix} \cos\omega_{x0} & \sin\omega_{x0} \\ -\sin\omega_{x0} & \cos\omega_{x0} \end{bmatrix} \begin{bmatrix} x_{i+1,j}^R(k) \\ x_{i+1,j}^I(k) \end{bmatrix} + \\ \begin{bmatrix} \cos\omega_{y0} & \sin\omega_{y0} \\ -\sin\omega_{y0} & \cos\omega_{y0} \end{bmatrix} \begin{bmatrix} x_{i,j+1}^R(k) \\ x_{i,j+1}^I(k) \end{bmatrix} + \\ \begin{bmatrix} \cos\omega_{y0} & -\sin\omega_{y0} \\ \sin\omega_{y0} & \cos\omega_{y0} \end{bmatrix} \begin{bmatrix} x_{i,j-1}^R(k) \\ x_{i,j-1}^I(k) \end{bmatrix} + \begin{bmatrix} \lambda^2 \\ 0 \end{bmatrix} u_{ij} \end{array} \right\}$$

where

$$G_x = \begin{bmatrix} \cos\omega_{x0} & -\sin\omega_{x0} \\ \sin\omega_{x0} & \cos\omega_{x0} \end{bmatrix} \text{ and } G_y = \begin{bmatrix} \cos\omega_{y0} & -\sin\omega_{y0} \\ \sin\omega_{y0} & \cos\omega_{y0} \end{bmatrix}$$

are Givens rotations. Now defining the vector

$$\boldsymbol{x}_{i,j}(k) = \begin{bmatrix} x_{i,j}^R(k) \\ x_{i,j}^I(k) \end{bmatrix}$$

we can write

$$\boldsymbol{x}_{i,j}(k+1) = \frac{1}{4+\lambda^2}\left\{ \begin{array}{l} G_x \boldsymbol{x}_{i-1,j}(k) + G_x^t \boldsymbol{x}_{i+1,j}(k) + \\ G_y^t \boldsymbol{x}_{i,j+1}(k) + G_y \boldsymbol{x}_{i,j-1}(k) + \lambda^2 u_{ij} \end{array} \right\} \quad (4)$$

Since G_x and G_y are Givens rotations note that

$$\left|G_x \boldsymbol{x}_{i,j+1}(k)\right| = \left|\boldsymbol{x}_{i,j+1}(k)\right|, \left|G_x^t \boldsymbol{x}_{i+1,j}(k)\right| = \left|\boldsymbol{x}_{i+1,j}(k)\right| \quad (5.a)$$

$$\left|G_y^t \boldsymbol{x}_{i,j+1}(k)\right| = \left|\boldsymbol{x}_{i,j+1}(k)\right|, \left|G_y \boldsymbol{x}_{i-1,j}(k)\right| = \left|\boldsymbol{x}_{i-1,j}(k)\right| \quad (5.b)$$

where $\left|\ \right|$ denotes the Euclidean norm. Now using (4) and (5) we can write

$$\begin{aligned} \left|\boldsymbol{x}_{i,j}(k+1)\right| &= \frac{1}{4+\lambda^2}\left|G_x \boldsymbol{x}_{i-1,j}(k) + G_x^t \boldsymbol{x}_{i+1,j}(k) + G_y^t \boldsymbol{x}_{i,j+1}(k) + G_y \boldsymbol{x}_{i,j-1}(k) + \lambda^2 u_{ij}\right| \\ &\le \frac{1}{4+\lambda^2}\left\{\left|G_x \boldsymbol{x}_{i-1,j}(k)\right| + \left|G_x^t \boldsymbol{x}_{i+1,j}(k)\right| + \left|G_y^t \boldsymbol{x}_{i,j+1}(k)\right| + \left|G_y \boldsymbol{x}_{i-1,j}(k)\right| + \left|\lambda^2 u_{ij}\right|\right\} \\ &= \frac{1}{4+\lambda^2}\left\{\left|\boldsymbol{x}_{i-1,j}(k)\right| + \left|\boldsymbol{x}_{i+1,j}(k)\right| + \left|\boldsymbol{x}_{i,j+1}(k)\right| + \left|\boldsymbol{x}_{i-1,j}(k)\right| + \left|\lambda^2 u_{ij}\right|\right\} \\ &\le \frac{1}{4+\lambda^2}\left\{4+\lambda^2\right\} = 1 \end{aligned}$$

We can conclude that if the input image pixel values and the initial states satisfy

$$\left|u_{i,j}\right| \le 1, \left|x_{ij}^R(0)\right| \le 1, \left|x_{ij}^I(0)\right| \le 1 \text{ for } i = 1,.....,M;\ j = 1,.....,N$$

then the real and imaginary parts of the output image pixel values are kept in the same range during the whole processing, i.e.,

$$\left|x_{ij}^R(k)\right| \le 1, \left|x_{ij}^I(k)\right| \le 1 \text{ for } i = 1,.....,M;\ j = 1,.....,N$$

Our 2-D Gabor-type filter digital circuit is given in Fig.1 [5].

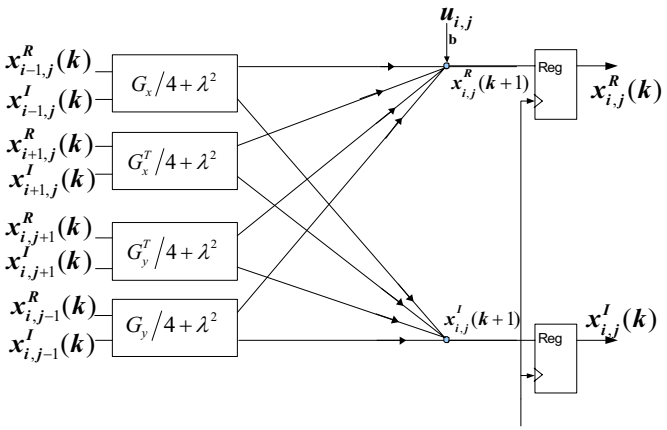

Figure 1. 2-D Gabor-type filter digital circuit

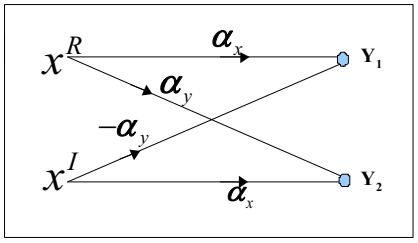

Figure 2. Butterfly structure

where

$$\alpha_x = \frac{\cos\omega_{x0}}{4+\lambda^2},\ \beta_x = \frac{\cos\omega_{y0}}{4+\lambda^2},\ \alpha_y = \frac{\sin\omega_{x0}}{4+\lambda^2}, \beta_y = \frac{\sin\omega_{y0}}{4+\lambda^2}, b = \frac{\lambda^2}{4+\lambda^2}.$$

III. SIMULATION AND ERROR ANALYSIS

The Gabor-type filter illustrated in Fig. 1 can be formed from a simple butterfly structure realizing the Givens rotation which is shown in Fig. 2. and are defined as

$$Y_1 = x^R\alpha_x - X^I\alpha_y$$
$$Y_2 = X^R\alpha_y + X^I\alpha_x \quad (9)$$

In (9), the values of x^R, x^I are between -1 and 1 and the coefficients of $\alpha_x, \beta_x, \alpha_y, \beta_y$ and b are between -0.25 and 0.25. Filter coefficients are changed according to ω_x, ω_y and λ. Filter coefficients that are obtained for different values of ω_x, ω_y, and λ, shown in Table I.

TABLE I. FILTER COEFFICIENTS

ω_{x0}	ω_{y0}	λ	α_x	β_x	α_y	β_y	b
30^0	60^0	0,5	0,2037707	0,1176471	0,1176471	0,2037707	0,0588235
90^0	90^0	0,5	0	0	0,2352941	0,2352941	0,0588235
1^0	33^0	0,5	0,2352583	0,1973343	0,0041065	0,1281504	0,0588235
30^0	60^0	0,2	0,2143627	0,1237624	0,1237624	0,2143627	0,009901
90^0	90^0	0,2	0	0	0,2475248	0,2475248	0,009901
1^0	33^0	0,2	0,2474871	0,2075917	0,0043199	0,1348116	0,009901
30^0	60^0	0,1	0,2159664	0,1246883	0,1246883	0,2159664	0,0024938
90^0	90^0	0,1	0	0	0,2493766	0,2493766	0,0024938
1^0	33^0	0,1	0,2493386	0,209145	0,0043522	0,1358202	0,0024938

TABLE II. RELATION BBETWEEN λ AND b

λ	$b = \lambda^2/4 + \lambda^2$						
	floating	Q1.3	Q1.7	Q1.11	Q1.13	Q1.15	Q1.19
0,5	0,0588235	0	0,062500	0,058593	0,058837	0,058837	0,058822
0,4	0,0384615	0	0,039062	0,038574	0,038452	0,038452	0,038461
0,3	0,0220049	0	0,023437	0,021972	0,021972	0,022003	0,022005
0,2	0,0099010	0	0,007812	0,009765	0,009887	0,009887	0,009901
0,1	0,0024938	0	0	0,002441	0,002441	0,002502	0,002492
0,01	0,0000250	0	0	0	0	0,000030	0,000024
0,001	0,0000002	0	0	0	0	0	0

From Table I, we can see that λ has a small effect on $\alpha_x, \beta_x, \alpha_y, \beta_y$. On the other hand its effect basically is on the coefficient b. The relation between λ and b is shown in Table II.

In fixed-point arithmetic word length of the coefficient of b must be at least 16-bit for low-bandwidth (λ), as shown in Table II. This format consists of two fields, a sign field is found at first and followed by a fractional field. In this case the value of this fixed-point number is representable in the range [-1, 1) with precision 0,000030517578125. We called this fixed-point format Q1.15.

Word length of the coefficient of $\alpha_x, \beta_x, \alpha_y, \beta_y$ can be altered depending on ω_{x_0}, ω_{y_0}. Word lengths of α_x, β_x, α_y, β_y are determined by the values defined in Table I. In order to implement the ω_{x_0}, ω_{y_0} values of Gabor-type filter with 1 degree of precision $\alpha_x, \beta_x, \alpha_y, \beta_y$ must be at least 12 bits. The 12-bit data format consist of 1 bit sign part and 11-bit fractional part. In this format results are within [-1, 1) and with precision 0,00048828125.

State variables defined as Q1.15 16-bit singed fixed-point numbers.

If fix-point arithmetic is used instead of floating point arithmetic for the Gabor-type filter architecture shown in Fig 1 and Fig 2, error will be as shown in Table III. Error criterion that was used is given in (10).

$$PSNR = 10\log_{10}\left(\frac{MN\max(I_{flt})^2}{\sum_{r=0}^{M-1}\sum_{c=0}^{N-1}\left[I_{fix}(r,c) - I_{flt}(r,c)\right]^2}\right) \quad (10)$$

TABLE III. PSNR VALUES

ω_{x0}	ω_{y0}	λ	PSNR(Q1.15)		PSNR(Q1.7)	
			x^R	x^I	x^R	x^I
30^0	60^0	0.5	85,96	88,07	37,09	39,02
90^0	90^0	0.5	90,21	91,91	40,58	43,71
1^0	33^0	0.5	85,19	86,29	36,67	37,00
30^0	60^0	0.1	86,23	87,36	37,60	38,60
90^0	90^0	0.1	88,80	90,49	43,40	43,91
1^0	33^0	0.1	85,08	86,40	37,14	38,32
30^0	60^0	0.01	86,04	87,23	38,20	38,36
90^0	90^0	0.01	90,65	93,20	43,18	43,91
1^0	33^0	0.01	85,55	85,42	37,30	38,20

Results of the MATLAB simulation of the Gabor-type filter using Q1.15 fix-point format is given in Fig.3.

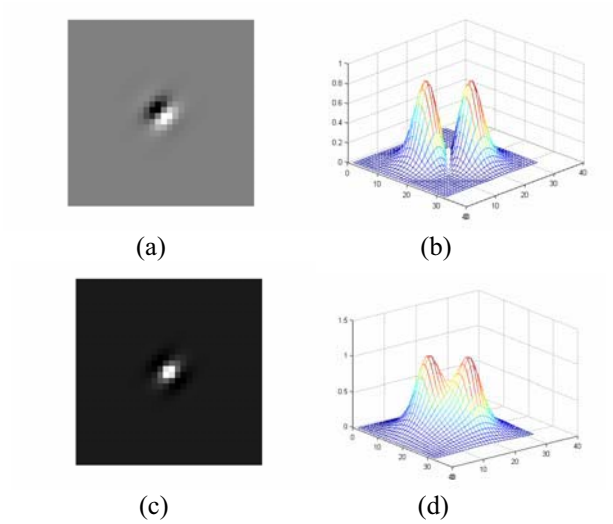

Figure 3. In Q 1.15 format for $\omega_{x_0} = \omega_{y_0} = \pi/4$ and $\lambda = 0.5$ (a) impulse response of odd Gabor filter, (b) frequency response of odd Gabor filter, (c) impulse response of even Gabor filter, (d) frequency response of even Gabor filter

As the final pixel values of the output in CNN are independent of initial conditions and computation errors, truncating is used instead of rounding.

IV. FPGA CORE

In order to form Gabor-type filter structure shown in Fig 1, 16 multipliers and 15 adders are needed. The structure is formed by the repeating blocks shown in Fig.2.

FPGA implementation of this block gives the same results as its MATLAB simulation. Block diagram of Fig.2 is depicted in Fig.5.

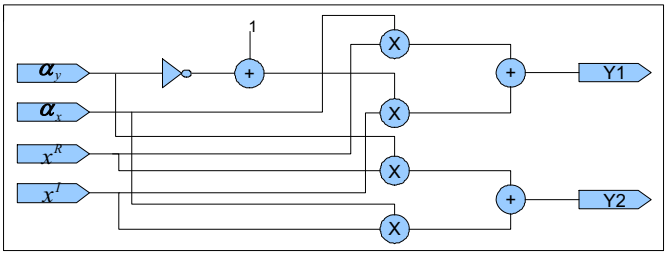

Figure 4. Butterfly Structure

2-D Gabor-type filter is shown in Fig. 5. is designed with register, adder and butterfly structure. The butterfly structure has four Mult18x18, two adders and one obtainer two's complement of input. This architecture was written with VHDL and synthesized by Xilinx ISE 8.2i. After synthesizing this VHDL code, tests were done with MODELSIM.

When this process unit is synthesized, placed and routed for XC3S400 FPGA, a 70-MHz clock rate can be achieved. This architecture covers 219 slices (%6), 24 slice flip-flops (%3), and 16 Mult18x18 (%100) on FPGA. An image of dimension 60 x 60 can be processed without using any external RAM only with the block RAM of XC3S400 FPGA.

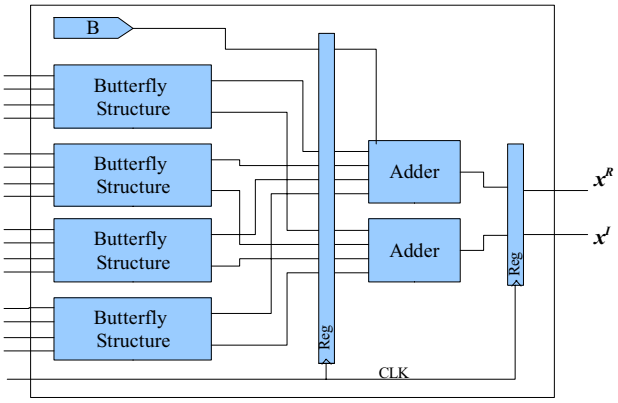

Figure 5. 2-D Gabor-type CNN filter processing unit

V. CONCLUSION

In this paper we presented a new digital realization for the implementation of 2-D Gabor convolution kernel. The complex valued kernel A of Gabor-type filter is separated into four parts. Forward Euler integration algorithm is used to simulate the CNN equation with the optimal step-size given in [3].

The new architecture reduces the complexity of the implementation of 2-D Gabor-type filter achieving a high clock rate using a low cost FPGA.

REFERENCES

[1] B. E. Shi, "Gabor-Type Image Filtering with Cellular Neural Networks," IEEE International Symposium on Circuits and Systems Proceedings (ISCAS'96), vol. 3, pp. 558-561, May 1996.

[2] O. Y. H. Cheung, Philip H. W. Leong, Eric K. C. Tsang and B. E. Shi, "A Scalable FPGA Implementation of Cellular Neural Networks for Gabor-type Filtering," International Joint Conference on Neural Networks, 2006. IJCNN '06, pp. 15-20, Vancouver, BC, Canada, July 16-21, 2006

[3] E. Saatci, "On the Optimal Choice of Integration Time-step for Raster Simulation of a CNN for Gray Level Image Processing," IEEE International Symposium on Circuit and Systems (ISCAS'02), pp. 625-628, Arizona, USA, 26-29 May 2002

[4] L. Chua and T. Roska, *Cellular Neural Networks and Visual Computing: Foundations and Applications*, Cambridge, Cambridge University Press, 2002.

[5] E. Saatci, "Image Processing Using Cellular Neural Networks," Ph.D. Dissertation, London South Bank University, London, U.K., Sept. 2003.