A Method of Quick Edge Detection Based on Zynq

Rong Xie
Department of information, Beijing University of Technology
Beijing, China
xierong8989@163.com

Xiaoqin Feng
Department of information, Beijing University of Technology
Beijing, China
fengxqinx@163.com

***Abstract*—Most of existing image processing algorithms have poor real-time performance in embedded device applications and occupy too much software resources. In this paper, a fast edge detection method is proposed based on Zynq platform. The xfOpenCv acceleration library provided by Xilinx is used to implement the edge detection algorithm. Specifically, the function operation process is accelerated in the FPGA through the acceleration constraint of a specific function implemented in the xfOpenCv library. In the implementation process, the SDx development environment is also used to simplify the function hardware acceleration development process. The experimental results show that our edge detection method takes less time than other methods, and has better real-time performance with the basically same detection results.**

Keywords—edge detection, Zynq, hardware acceleration

I. INTRODUCTION

Image processing technology originated in 1920s. Early image processing is mainly based on human objects to improve the quality of images and meet the visual effects of people. In the 1960s, with the development of electronic computer technology, people began to use computers to process image information. With the deepening of research, people began to study how to use computers to analyze and interpret images [1]. Its process resembles human visual understanding of the external world.

With the rapid development of electronic technology, more and more traditional digital image processing system can be realized through embedded system. Embedded system has a series of advantages such as small size, low cost, low power consumption and good reliability. The embedded image processing system based on embedded technology has been widely used in industrial testing, machine vision, aerospace, military guidance, biomedicine, public safety, Car driving auxiliary and other fields [2]. At present, embedded processors used in image processing mainly include ARM, DSP and FPGA. However, the existing embedded image processing solutions based on a single ARM processor, DSP or FPGA can no longer meet the requirements of high-performance applications. Using multiple chips combined solution will often result in complex hardware architecture, difficult development, system instability and other issues. In response to the above issues, the paper uses Xilinx Zynq all-programmable platform. The platform is the first heterogeneous chip that tightly integrates the high-performance ARM Cortex A9 hardcore and the programmable logic FPGA. Through this combination, Zynq not only has the transaction management, operating system and other advantages in ARM processor, but also has parallel processing, the dynamic reconfiguration and other advantages in FPGA [3, 4, 5].

This paper designs a fast edge detection method under the Zynq platform using the xfOpenCv kernel proposed by Xilinx. The xfOpenCv is a library designed in Xilinx SDx development environment. xfOpenCv is an improvement on OpenCv made by Xillinx for embedded environments. Its functions are mostly similar in functionality to their OpenCV equivalent. It's intended for application developers using Zynq-7000 All Programmable SoC and Zynq UltraScale+ MPSoC devices. It provides a software interface for computer vision functions accelerated on an FPGA.

II. THE METHOD OF EDGE DETECT

A. Canny edge detection algorithm

This paper uses canny edge detection algorithm as an example. Canny edge detection can be divided into four steps: image smoothing, gradient and direction calculations, non-maximum suppression, detect and connect edges. The first step: image smoothing. The Canny algorithm uses a two-dimensional Gaussian function to smooth the image, shown in Equation (1).

$$G(x,y)=\frac{1}{2\pi\ \sigma^2}\exp(-\frac{x^2+y^2}{2\sigma^2}) \qquad (1)$$

In Equation (1), σ is a Gaussian filter parameter, which controls the degree of smoothness. The filter with smaller σ has high accuracy of positioning but low signal to noise ratio. The filter with large σ is just the opposite. Therefore, we need to select Gaussian filter parameters as required. The second step: gradient and direction calculations. The Canny algorithm uses a first-order differential operator to calculate the gradient magnitude and gradient direction at each point of the smoothed image to obtain the corresponding gradient magnitude image G and gradient direction image θ. The partial derivatives at points (i,j) in both directions are $G_x(i,j)$ and $G_y(i,j)$, shown in Equation (2) and (3).

$$G_x(i,j)=(I(i,j+1)-I(i,j)+I(i+1,j+1)-I(i+1,j))/2 \qquad (2)$$

$$G_y(i,j) = (I(i,j) - I(i+1,j) + I(i,j+1) - I(i+1,j+1))/2 \quad (3)$$

At this point, the gradient magnitude and gradient direction at point (i,j) are $G(i,j)$ and $\theta(i,j)$, shown in Equation (4) and (5).

$$G(i,j) = \sqrt{G_x^2(i,j) + G_y^2(i,j)} \quad (4)$$

$$\theta(i,j) = \arctan\left(\frac{G_x(i,j)}{G_y(i,j)}\right) \quad (5)$$

The third step: non-maximum suppression. In order to accurately position the edges, the ridge band in the gradient-magnitude image G must be refined to only retain the local maxima of the amplitude, ie non-maximal suppression (NMS). The Canny algorithm interpolates along the gradient direction $\theta(i,j)$ in the neighborhood of 3 × 3 with the point (i,j) as the center in the gradient image G. If the gradient value $G(i,j)$ at the point (i,j) is larger than the two adjacent interpolation values in the direction of $\theta(i,j)$, the point (i,j) is marked as a candidate edge point. Otherwise, it is marked as a non-edge point. The fourth step: detect and connect edges. The Canny algorithm uses double-threshold method to detect and connect the final edge from candidate edge points. The double threshold method first selects the high threshold T_h and the low threshold T_l, and then starts scanning the image. It is detected any pixel point (i,j) marked as a candidate edge point in the candidate edge image. If the point (i,j) gradient magnitude $G(i,j)$ is higher than the high threshold T_h, the point is considered to be an edge point. If the point (i,j) gradient magnitude $G(i,j)$ is lower than the high threshold T_l, the point is considered not to be an edge point. For pixel points whose gradient amplitude is between the two thresholds, they are regarded as suspected edge points and need to be further judged according to the connectivity of the edges. If there are edge points in the adjacent pixels of the pixel point, the point is also considered to be an edge point. Otherwise, it is considered to be a non-edge point [6, 7, 8]. Through the four-step processing of the picture by the canny algorithm, the edge of the picture can be well detected. On the other hand, using the Canny operator edge detection algorithm, the detection result image can also respond better to the weaker edges.

B. The advantages of edge detection base Zynq

In the field of embedded image processing, due to the constraints of system architecture, memory bandwidth and other factors, it is difficult for traditional single-chip microcomputer systems to meet the requirements of throughput and computer performance at the same time. In 2011, Xilinx introduced the Zynq-7000 series of scalable platforms that integrate programmable logic with the dual-core ARM A9 processing system. With integrated ARM processors, Zynq-7000 leverages existing embedded resources. The integrated FPGA resources allow the Zynq-7000 to provide high-performance computing solutions [9]. Based on the significant advantages of the ZYNQ platform, the video image processing system under the embedded platform mainly includes a programmable logic (PL) and a processing system (PS). Part of the PL is mainly responsible for the control logic of each module, including the DMA transmission of video image data and HD display. It can achieve hardware acceleration of video image processing algorithms in the high-level synthesis tools. PS part is mainly responsible for running Linux system on ARM Cortex A9 [10].

In the general Zynq development process, the OpenCv library needs to be ported to the Linux operating system running in the arm kernel in Zynq. Running image processing programs in Zynq need to cross-compile and call the ported OpenCv library file to execute the program. In practical applications, there is a problem that image processing methods based on the ARM platform are too slow in running complex algorithms [11, 12]. XfOpenCv library is Xilinx optimizes OpenCv with FPGA. XfOpenCv can be used in the device application development of Zynq-7000 All Programmable SoC and Zynq UltraScale+ MPSoC. xfOpenCV library has been designed to work in the SDx development environment, and provides a software interface for computer vision functions accelerated on an FPGA device. Figure 1 shows the architecture of xfOpenCv in Zynq.

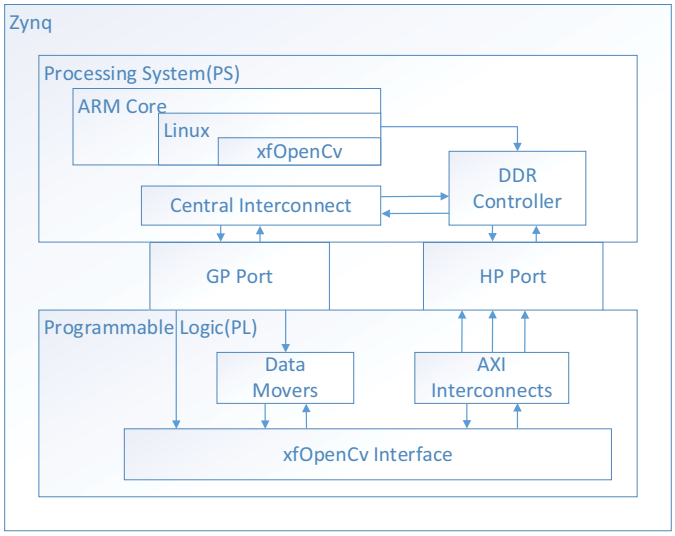

Figure. 1 The architecture of xfOpenCv in Zynq

III. THE DESIGN OF A FAST EDGE DETECTION METHOD

This paper implements a fast edge detection method by Xilinx xfOpenCv libraries. The implementation process applies to the high-level synthesis tools Vivado HLS and Xilinx Zynq SOC all-programmable processing platform, Vivado image video library, and converts OpenCv to RTL through the advanced synthesis tool Vivado HLS. Hardware acceleration of

OpenCv program algorithms is implemented on Xilinx Zynq SoC series all-programmable processing platforms. Take the Canny edge detection as an example.

A. The implementation of edge detection method

This paper takes the Canny algorithm as examples. It is based on Xilinx xfOpenCv library design and implementation. In this method, the Mat data format needs to be converted to the hls::stream data format. The edge detection process of the Canny algorithm is divided into four steps: image filtering, gradient calculation, non-maximal suppression, and connection edges. After processing, it convert the hls::stream data type to the Mat data format. The processing flow of Canny algorithm figure shown in Figure 2. The data format of the input image needs to be converted. The data format of the picture read from by OpenCv is Mat data type. The Mat data type is converted into xf::Mat type data through the copy operation. The xf::Mat data format is an intermediate format that prepares for the next data format conversion to hls::stream. The data of xf::Mat type needs to assign each row and column data to the data of hls::stream type. The assignment process uses two loop processes: row loop and column loop. The two loop processes are optimized using the existing optimization method in HLS. There are two optimizations for the loop: the pipeline constraint and the dataflow constraint. The pipeline constraint causes the loop in the function to be pipelined. Pipelined execution loops increase the parallelism of operations. Because there is a data transfer relationship between the row loop and the column loop, the dataflow constraint is added between the two loops so that different loop bodies execute in parallel and the throughput of the data can be improved. Converting Mat data to the hls::steam data format allows the program to achieve optimal performance during execution. After the data format conversion is complete, the canny algorithm of edge detection starts the edge detection part of the calculation. The first step is the Gaussian filtering process. In the Gaussian filtering process, we need to matrix convolute for each pixel of the image. The matrix convolution can be divided by ARRAY_PARTITION constraint. In Gaussian filtering calculation, the kernel number of Gaussian convolution is three, so the matrix can be divided into three parts by using ARRAY_PARTITION constraint. The parallel bandwidth and the computing power in the calculation process of the divided matrix. On the other hand, the #pragma HLS RESOURCE variable=buf core=RAM_S2P_BRAM constraint adds the matrix buf to the RAM. It controls the delay of generating BRAM to ensure the correct timing in the matrix operation. The second step is to use the Sobel algorithm to calculate the image gradient, which is the same as the Gaussian filtering process. The gradient of the calculated graph also involves matrix convolution operations. It also uses ARRAY_PARTITION constraint to accelerate the convolution of the matrix. The third step is non-maximum suppression. The specific operation process is to perform a convolution operation with a 3*3 filter kernel and the input image. The calculation process also involves a convolution operation. The ARRAY_PARTITION constraint is also used to accelerate the matrix convolution operation speed. The fourth step is the process of connecting the edges. It uses the high and low thresholds set by the user to determine if the pixel is an edge point. The computation process optimizes the loop process through pipeline constraints, which improves parallel computing capabilities. After the Canny algorithm is completed, the image data format is converted from hls::stream to xf::mat, and finally restored to the Mat type. The entire computational acceleration process utilizes the parallelism of the FPGA. It accelerates the matrix convolution and the cyclic process through parallel operations that improves the speed of image processing and the real-time performance in real-time video processing.

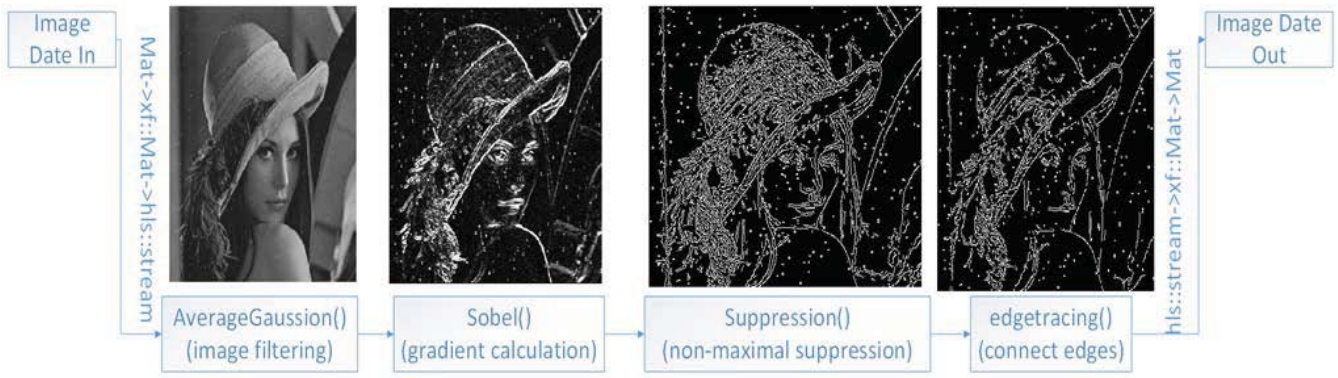

Fig. 2 The processing flow of canny algorithm

B. The contrast experiment

This paper takes edge detection of Canny algorithm as an example, and compares the time spent and the quality of the results in the three cases. The experimental environment configured on the PC is Ubuntu16.4 and OpenCv3.4.1. The development board used in the experiment is the ZUC102 development board under the Zynq series of Xilinx. The development board experimental environment is to transplant the Linux system on the PS side. Cross-compilation of OpenCv source code for the development board support library files ported to the development board, so that the development board can support cross-compiled OpenCv program. The experiment needs to establish a Zynq-based development project in SDx, and write the edge detection source code. The most critical steps is to put hardware-accelerated parts of code into hardware for acceleration. The part that can be accelerated in this experiment is the Canny algorithm part. The functions implemented in the hardware call FPGA resources at runtime to achieve parallel acceleration, improve program execution

speed, and save software resources. The results of edge detection in three environments are shown in Figure 3.

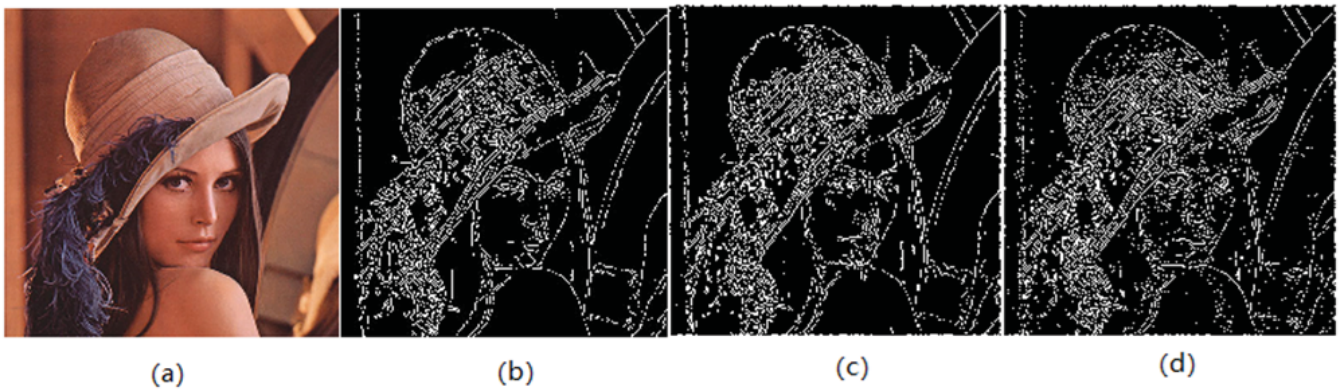

Fig. 3 The result of edge detection in three environments

Figure 3 (a) the original image input. (b) the result of PC environment. (c) the result of unaccelerated in the development board environment. (d) the result of accelerated in the development. The edge detection process in the three environments sets the same high and low thresholds, and the result of the edge detection is basically the same. Programs that are accelerated using the xfOpenCv library output the same results as the general case. Comparing program runs takes time in the same situation where edge detection results. Experiments tested the time-consuming of edge detection for twenty different pictures. Time-consuming in the three environments are shown in Table 1.

TABLE I. TIME-CONSUMING IN THE THREE ENVIRONMENTS

Enviement	**PC**	**OpenCv of board**	**Ours**
Highest time-consuming(s)	0.009230	0.072895	0.005337
Lowest time-consuming(s)	0.006940	0.060540	0.004096
Average time-consuming(s)	0.008030	0.066191	0.004170

It can be concluded that the accelerated algorithm has a significant reduction in time-consuming. The use of xfOpenCv to accelerate the canny algorithm for edge detection has some advantage in time-consuming than other environments.

IV. SUMMARY

In this paper, a method of quick edge detection is proposed based on Zynq By using the xfOpenCv library. Our method uses existing acceleration methods to accelerate the edge detection process, and utilizes the parallel computing capabilities of FPGAs to speed up the edge detection process. Compared with the operation of the edge detection algorithm in other environments, our method can obtain the detection results that are basically consistent with the general case. However, it has greatly reduced the computational time.

Further studies beyond this work include improving the accuracy of edge detect result and test the performance of other algorithm in our acceleration environment, and applying accelerated thinking to other applications.

REFERENCES

[1] Pauwels K, Tomasi M, Alonso J D, et al. A Comparison of FPGA and GPU for Real-Time Phase-Based Optical Flow, Stereo, and Local Image Features. IEEE Transactions on Computers. Vol. 61 (2012) No. 7, p. 999-1012.

[2] Farabet C, Martini B, Akselrod P, et al. Hardware accelerated convolutional neural networks for synthetic vision systems IEEE International Symposium on Circuits and Systems. IEEE, 2010, p, 257-260.

[3] Appiah K, Hunter A, Dickinson P, et al. Accelerated hardware video object segmentation: From foreground detection to connected components labelling. Computer Vision & Image Understanding. Vol. 114 (2010) No. 11, p. 1282-1291.

[4] Honegger D, Oleynikova H, Pollefeys M. Real-time and low latency embedded computer vision hardware based on a combination of FPGA and mobile CPU. Ieee/rsj International Conference on Intelligent Robots and Systems. IEEE, 2014, p. 4930-4935.

[5] Savarimuthu, Rajeeth T, Kj, et al. Real-time medical video processing, enabled by hardware accelerated correlations. Journal of Real-Time Image Processing. Vol. 6 (2011) No. 3, p. 187-197.

[6] Wei-Bo Y, Du Z S. An improved Kirsch human face image edge-detection method based on canny lgorithm. International Conference on Consumer Electronics, Communications and Networks. IEEE, 2011, p. 4740-4743.

[7] Pan Q S, Zhang Y, Yang Z M, et al.Design and Implementation of Image Corner and Edge Detection System Based on Zynq. Computer Science. 2017.

[8] Xin G, Ke C, Hu X,. et al. An improved Canny edge detection algorithm for color image. IEEE International Conference on Industrial Informatics. IEEE, 2012, p. 113-117.

[9] Ahmad A M, Lukowicz P, Cheng J. FPGA based hardware acceleration of sensor matrix. ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct. ACM, 2016, p. 793-802.

[10] Dang V, Skadron K. Acceleration of Frequent Itemset Mining on FPGA using SDAccel and Vivado HLS. IEEE, International Conference on Application-Specific Systems, Architectures and Processors. IEEE, 2017, p. 195-200.

[11] Russell M, Fischaber S. OpenCV based road sign recognition on Zynq. IEEE International Conference on Industrial Informatics. IEEE, 2013, p. 596-601.

[12] Cortes A, Velez I, Irizar A. High level synthesis using Vivado HLS for Zynq SoC: Image processing case studies Design of Circuits and Integrated Systems. IEEE, p. 2017:1-6.