Proceedings of the 38th Chinese Control Conference
July 27-30, 2019, Guangzhou, China

Implementation of a FPGA-ARM-based Canny Edge Detection System

Zhaoyang Liu[1, 2], Fengshui Jing[1, 2], Junfeng Fan[1, 2], Zhe Wang[1, 2]

1. Institute of Automation, Chinese Academy of Sciences, Beijing 100190
E-mail: liuzhaoyang2017@ia.ac.cn

2. School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing 101408
E-mail: fengshui.jing@ia.ac.cn

Abstract: Canny edge detection is widely used in image processing. Conventional Canny edge detection is commonly realized in CPUs, GPUs or FPGA fabrics alone, but CPUs and GPUs are not suitable for low-power applications. And existing edge detection implementations on specific FPGA model have little compatibility with other models. In this paper, we propose an effective Canny edge detection system consist of FPGA and ARM-SoC. ARM core is embedded with Yocto Linux OS, and FPGA is in charge of image processing and VGA display. FPGA IP cores are designed to communicate through Avalon-ST Bridge. In the end, the system has been implemented and tested with a quantity of images. The testing results show the efficiency of the proposed system design scheme and the system can be based on to achieve more complex algorithm.

Key Words: FPGA+ARM, Canny edge detection

1 Introduction

Edge detection is a pre-process of many visual-based object detection tasks. Though new methods in detection such as deep neural network models can skip the part of explicitly extracting the edges of the inputs, edge detection is still used widely in all kinds of object detection and tracking tasks to produce high efficiency feature. For example, industrial practice still need precise edge detection for welding, object detection and autonomous driving.

A handful of operators have already been developed to detect edges, and these methods include Roberts, Prewitt, Sobel, Laplacian and Canny [1]. And there are different processor platforms to implement image processing systems, mature solutions including CPU, GPU, embedded device like ARM processor, DSP and FPGA [2].

CPUs and GPUs are common devices in image processing. CPUs have serial processing units running up to GHz with pipeline structures and their power consumption is around 100W. It is achievable for CPUs to process real-time images. For example, in paper [3] Ahmed Sherif Zekri1 achieves high speed single core CPU image processing with source-code restructuring. GPUs possess better parallel computing architecture thanks to thousands of graphical units. It is not hard for GPUs to achieve real-time detection tasks. However, GPUs usually run at lower clock frequency (around 1600MHz), and consume more power than a same level CPU. GPUs always work together with CPUs because they do not have their own operating system. CPU+GPU is the mainstream solution for image processing if the power and size are out of consideration.

Embedded devices like FPGAs also have parallel image processing ability. It has the advantage of flexible logic units, high integration degree and wide scope of implement range. Though FPGAs work at lower frequency around 200MHz, they can achieve real-time image processing by coded parallelism and pipeline structure. FPGAs do not need external memory for intermediate data and parameter storing, thus they do not need high bandwidth RAMs for data exchange. FPGAs are energy efficient. In [4], a FPGA based implementation of HW-CNN performs equally to a CPU with 16 threads in the aspect of performance per watt, outperforming a mobile GPU by 19 times and a single-threaded CPU by 3 times. But algorithm implementation on FPGA is time-consuming since engineers need to build complex algorithms at a circuit level.

In comparison, DSPs are weak in parallel computation, and yet used sometimes in image processing. ARM cores are low-power, portable, cheap solutions for image processing. However, they do not have as many transistors as CPUs and often work with battery in mobile devices, which limit their real-time image processing speed.

In the case of embedded image processing, CPU and GPU based implementation are no longer practical in size and in power. But it is also too overwhelming for single embedded device like ARM, DSP, and FPGA to achieve real-time processing and embed an operating system for further development.

To solve the problem of real-time embedded image processing, it is advisable to utilize the merits of heterogeneous systems. ARM+FPGA system has both good performance and flexibility in image processing due to high energy efficiency of FPGA and embedded operating system provided by ARM. Wenchao Liu et al. designed a moving object detection with Xilinx Zynq-7000 FPGA, with ARM processor inside [5]. Shreekant Sajjanar et al. also use this kind of system structure with modified background subtraction to detect moving object [6].

Previous FPGA image processing implementations [7] [8] tend to design the whole system with special designed data transfer interface, while Intel has its own data interface Avalon-ST for video processing. With this this standard, developers can wrap their algorithms and data with standard data packets and IP cores for more complex systems and faster FPGA fabric design.

All aspect considered, this paper proposes an image processing system to implement Canny edge detection on

*This work was supported by the National Natural Science Foundation of China under Grant No. U1813208 and 61573358.

ARM+FPGA device with Avalon-ST interface. The rest of the paper is arranged to introduce Canny edge detection acceleration method on FPGA, the design of system, experiment results and a brief comparison of performance on different platforms.

2 Canny Edge Detection in FPGA

Canny edge detection is a multi-filter detection method with balanced performance in edge continuity and signal to noise ratio, brought up by John Francis Canny in [9]. Discussed in [10], it has low probability of failing to mark edges or falsely marking non-edges; it has good edge position accuracy; and it has low probability of multiple responses to one edge.

The algorithm includes Gaussian smoothing, Sobel operation, non-maximum suppression, and hysteresis thresholding shown in Fig. 1. However, Canny's algorithm works mathematically, and this paper shows a digitized realization of Canny edge detection for FPGAs.

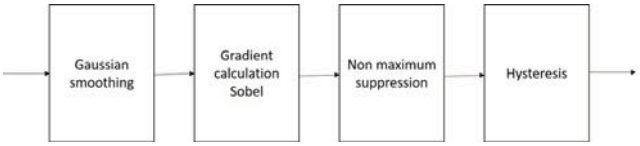

Fig. 1: Canny edge detection algorithm structure

To begin with the process, Gaussian smoothing filter is used for noise reduction in that Sobel operation is sensitive to noise. A 2D Gaussian filter (1) can be simplified digitally by convolution with a 5x5 kernel. Two kernels of different standard deviation δ are shown in Fig. 2. It is tested in [10] that a 5x5 sized kernel is enough for keeping a balance between filtering speed and accuracy. And the scaling weights can be rounded to powers of 2, so that division can be replaced by register shifting in FPGA. For example, division of 115 can be replaced by 128 which is a 7 bit right shift of the data.

$$G(x,y) = \frac{1}{2\pi\delta^2} e^{-\frac{x^2+y^2}{2\delta^2}} \tag{1}$$

$$\frac{1}{115}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} \quad \frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Fig. 2: Gaussian kernels of δ=1.4 and δ=1

The Sobel operator (2) is a discrete convolution operator for differentiation as in [12].

$$G_{Sobel-x} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}, G_{Sobel-y} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{2}$$

The convolution results are gradient magnitude components G_x and G_y of x and y direction. Then equation (3) and (4) are used to calculate the magnitude and direction of the gradient from two components.

$$G = \sqrt{G_x^2 + G_y^2} \tag{3}$$

$$\theta = \arctan\left(G_y / G_x\right) \tag{4}$$

Equation (3) can be modified to equation (5), so FPGA do not need to perform square operation and take the root of the sum. This simplification will save at least 16 multiplexor and 1369 logic units by synthesizer for taking the square root. In this way, only one adder is needed. It is also not easy for FPGA to calculate an inverse tangent. A precise calculation of FPGA uses Cordic algorithm. However, the algorithm is slow and takes up a lot of FPGA resources. So equation (4) is simplified to a classification of angle into 8 sectors shown in Fig. 3. Each sector ranges 45°, coded by the sign and magnitude comparison. The simplification results in a short FPGA pipeline, less logic unit usage, and less latency of the output.

$$G = |G_x| + |G_y| \tag{5}$$

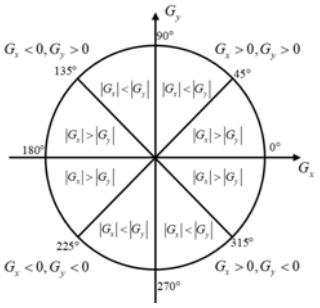

Fig. 3: Eight sectors of gradient angle

To improve the localization of thick edges, the next step of Canny edge detection is a suppression of non-maximum gradients. This is done by a local maximum search around a target pixel along its direction of gradient [13]. In practice, gradient interpolation is needed because pixels do not align with the sectors in Fig. 3. Example interpolation of 0°~45° and 45°~90° are shown in Fig. 4. If the pixel tested has greater gradient than two gradient interpolations on both along and opposite to its direction of gradient, the pixel is kept as an edge candidate. Otherwise, its gradient is set to 0. After the suppression, there will be no more edges thicker than 1 pixel.

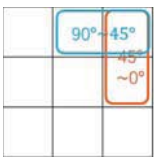

Fig. 4: Interpolation of gradients of 0°~45° and 45°~90

Finally, hysteresis thresholding is used to filter the weak edges and the strong edges for noise reduction and edge continuity preservation. In this process, edges are filtered to two sets $\{S_{strong} \mid x > t_{high}\}$ and $\{S_{weak} \mid t_{low} < x < t_{high}\}$. If break up point x_{break} of strong edges $\{S_{strong}\}$ appears,

points in $\{S_{weak}\}$ around x_{break} within an 8 neighbor range are added to $\{S_{strong}\}$ until there is no points in $\{S_{weak}\}$ to connect or points in $\{S_{strong}\}$ appear in the search range. This will improve edge final edge continuity as Fig. 5 shown. Moreover, multiple thresholds can be applied to connect edges for better performance.

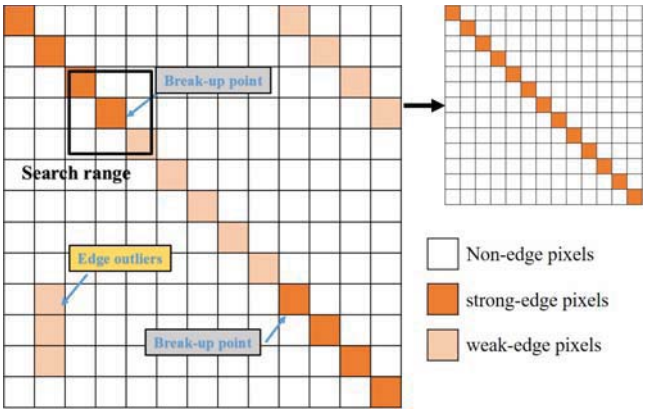

Fig. 5: searching for weak edge pixels around strong edge pixels for edge continuity

3 System Design

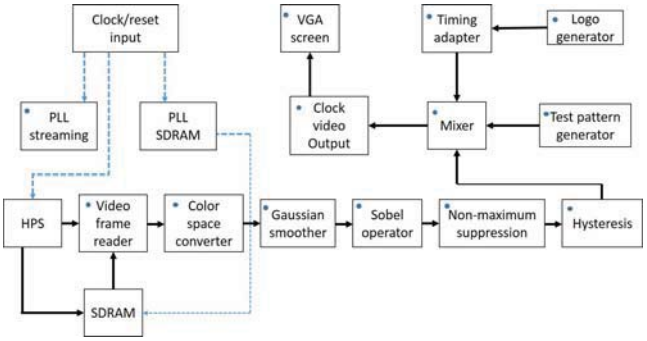

Fig. 6: Complete system structure (dots indicate the same clock input)

In this paper, we designed a FPGA+ARM system for Canny edge detection as Fig. 6. The system is based on DE1-SoC board with Altera Cyclone V FPGA and ARM Cortex-A9. Intel Altera Qsys is used to design the FPGA system. Image data is streamed from ARM side to FPGA side by Avalon bus, and FPGA computes the edges and accomplishes VGA output of the results. IP cores of Gaussian smoothing, Sobel operation, non-maximum suppression, and hysteresis thresholding are designed to work with Avalon bus. Verilog HDL is used to design these cores. These designed IP cores can work as general IP cores for later experiments, or work together as one core for edge detection. Other IP cores used in the system is from Intel Altera IP cores. The designed IP cores are fully compatible with the library IP cores, making the system modular and easy to be upgraded for complex tasks. And with the optimized Canny edge detection algorithm, the whole system can be easily modified for real-time video processing. The system is also low-power and portable. The RTL level system design is in Fig. 7 and IP core designed for Canny edge detection is in Fig. 8.

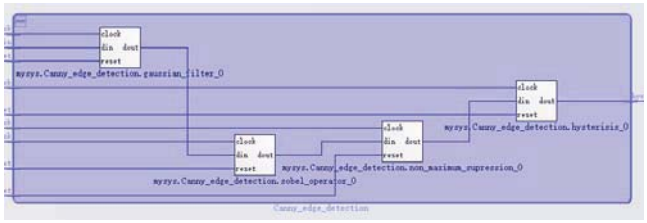

Fig. 8: Canny edge detection IP cores inside the subsystem

The HPS (Hard Processing System) system is composed of an ARM core and Yocto Linux running on it. HPS controls the behavior of the FPGA IP cores and FPGA SDRAM.

Original image data of RGB form is stored in Linux system memory. Image is transferred to FPGA SDRAM using a 128-bit HPS-to-FPGA AXI bridge. HPS sends control signal of the FPGA Frame Reader core via a 32-bit light-weight-AXI bridge to read the image from HPS memory to FPGA SDRAM. A single image is packed into a packet for Avalon video streaming interface.

Then the packet is processed in a pipeline fashion by the FPGA IP cores. Frame Reader streams packet to Color Space Converter core, transforming color frame to grayscale frame. Then greyscale core output packet is then processed sequentially by Gaussian smoothing core, Sobel core, non-maximum suppression core and hysteresis core.

The output of the hysteresis core is one of the input of Alpha Blending Mixer core. A static logo is generated from logo generator core. A timing adapter is used to match the timing of the logo with system timing and the output of the

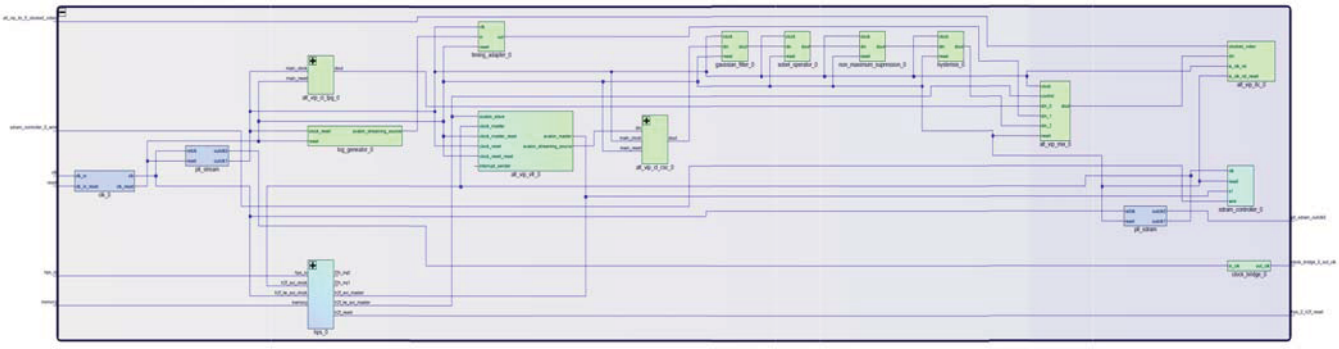

Fig. 7: RTL level system

timing adapter is sent to the mixer. The other input of the mixer is a black background generated from Test Pattern Generator core. At last, mixer output is transferred to Clock Video Output IP core so that a monitor from VGA port of FPGA fabric can show the result.

For clock domain, the default clock input is 50MHz, and two Integer-N type PLLs are used to produce the rest clocks. One Clock Bridge is used to conduit the clock to the VGA port. Control signal at light-weight AXI-bridge works at 50MHz clock rate. The data reading from SDRAM works at 120MHz. The image processing IP cores work at 180MHz.

Given the resolution of 1280*1024, a complete VGA scan needs 1688*1066 clock cycle. For a screen refreshing with 60Hz, the clock frequency of the VGA port is 108MHz as equation (6).

$$f_{ck} = 1688 \times 1066 \times 60 = 107964480Hz \approx 108MHz \quad (6)$$

4 Experiments and Comparison

According to the design, the HPS and FPGA systems are compiled. The flow summary of FPGA system of Quartus 16.1 is listed in Table 1

Table 1: FPGA flow summary

Logic utilization (in ALMs)	4,490 / 32,070 (14 %)
Total registers	9259
Total pins	386 / 457 (84 %)
Total virtual pins	0
Total block memory bits	381,000 / 4,065,280 (9 %)
Total DSP Blocks	6 / 87 (7 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserial-izers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	2 / 6 (33 %)
Total DLLs	1 / 4 (25 %)

We can see that the system only takes a small portion of the chip resource, indicating that more complex algorithm can be realized on this chip.

Then the system is deployed to DE1-SoC board for testing with 640*480 resolution input. Original images and test results are listed in Fig. 9 and Fig. 10. Images of streets are tested. The boundaries of arrows and zebra crossings are clearly detected and the edges have good continuity.

Since the ARM side data transfer only relies on the SDRAM read and write speed, and the final VGA output is sampled from the edge detection outputs with 60Hz. The Canny edge detection part of our system is compared in the aspect of speed against DSP system results from [14], CPU, GPU and another FPGA system results from [15], shown in Table 2. Though GPU can achieve the same level of performance, it has a clock speed 8 times over our FPGA fabric. DSP and CPU are also able to finish edge detection with in 30ms, but clearly they do not fit well with the task as FPGAs and GPU. Two FPGA system perform equal considering the clock rate difference and image size difference, since that pipeline processing guarantees one output at a single clock cycle. Our system can achieve the same detection time if we are given the same clock rate, but there is no point to accelerate the speed just by raise clock rate. And with the same performance, FPGA solution in [15] does not build the whole system with an ARM-Linux+FPGA-IP structure.

Table 2: Canny edge detection on different platforms

Platform	Device model	Image size	Clock speed (MHz)	Edge detection time (ms)
DSP	C64plus	768x288	600	24.27
CPU	Intel Core2 Duo E6600	512x512	2400	30
GPU	GeForce GTX 580	512x512	1540	2.11
FPGA in [15]	Arria V	512x512	242	1.1
FPGA in our system	Cyclone V	640x480	180	1.7

(a) (b)

Fig. 9: Original image

(a) (b)

Fig. 10: system detected edges

5 Conclusion and Future Work

This paper has realized a Canny edge detection system. The whole system utilized strong portability of ARM processor and high efficiency of FPGA fabric. IP cores are designed with Avalon ST standard bridges, compatible with Intel IP cores. The processing speed is high with low power consumption, and at the same time outputs are good in edge continuity and noise suppression.

Future works can be done to improve the edge detection precision and speed. Multi-threshold and adaptive-threshold method could be applied for better edge continuity. The system can be faster with multiple FPGA pipelines working at the same time. The input video signal can be demultiplexed in the time domain and processed respectively by different pipelines since that there is plenty of resource left in the FPGA fabric for multiple pipelines. And work can be done to improve fast load of the FPGA pipeline using ARM core to pre-process the data in order to reduce the output

latency. After the edge detection, the output can be used in real-time image segmentation, object detection and so on.

Furthermore, the system provides design example for FPGA+ARM image processing. Given the parallel computation ability of FPGA and operating system of ARM-Linux, we can speed up complex algorithm in the same fashion.

References

[1] Ganesan P, Sajiv G. A comprehensive study of edge detection for image processing applications[C]// 2017 4th International Conference on Innovations in Information, Embedded and Communication Systems. IEEE, 2017.

[2] Asano S , Maruyama T , Yamaguchi Y . Performance comparison of FPGA, GPU and CPU in image processing[C]// International Conference on Field Programmable Logic & Applications. IEEE, 2009.

[3] Zekri, Sherif A. Optimizing image spatial filtering on single CPU core [J]. Multimedia Tools and Applications, 2018, 77(1):251–281

[4] Bettoni M, Urgese G, Kobayashi Y, et al. A Convolutional Neural Network Fully Implemented on FPGA for Embedded Platforms [C]// 2017 New Generation of CAS (NGCAS). IEEE, 2017.

[5] Wenchao L , He C , Long M . Moving object detection and tracking based on ZYNQ FPGA and ARM SOC [C]// International Radar Conference. IET, 2016.

[6] S. Sajjanar, S. K. Mankani, P. R. Dongrekar, N. S. Kumar, Mohana and H. V. R. Aradhya, Implementation of real time moving object detection and tracking on FPGA for video surveillance applications[C]//2016 IEEE Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER), Mangalore, 2016.

[7] Hsu Y P, Miao H C, Tsai C C. FPGA implementation of a real-time image tracking system[C]// Sice Conference. 2010.

[8] Wu J, Sun J, Liu W. Design and implementation of video image edge detection system based on FPGA[C]// International Congress on Image & Signal Processing. IEEE, 2010.

[9] Canny J F. Finding Edges and Lines in Images[R]. Massachusetts Institute of Technology Cambridge Artificial Intelligence Lab, 1983.

[10] Canny J. A Computational Approach to Edge Detection [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2009, PAMI-8(6):679-698.

[11] Gentsos C, Sotiropoulou C L, Nikolaidis S, et al. Real-time canny edge detection parallel implementation for FPGAs[C]//Electronics, Circuits, and Systems (ICECS), 2010 17th IEEE International Conference on. IEEE, 2010: 499-502.

[12] He W, Yuan K. An improved Canny edge detector and its realization on FPGA[C]//Intelligent Control and Automation, 2008. WCICA 2008. 7th World Congress on. IEEE, 2008: 6561-6564.

[13] Neubeck A, Van Gool L. Efficient Non-Maximum Suppression[C]// International Conference on Pattern Recognition. IEEE Computer Society, 2006:850-855.

[14] Musoromy Z, Bensaali F, Ramalingam S, et al. Comparison of real-time DSP-based edge detection techniques for license plate detection[C]// International Conference on Information Assurance & Security. IEEE, 2010.

[15] Possa P R, Mahmoudi S A, Harb N, et al. A Multi-Resolution FPGA-Based Architecture for Real-Time Edge and Corner Detection [J]. IEEE Transactions on Computers, 2014, 63(10):2376-2388.