Development of Fixed-point Canny Edge Filter Operation for High-level Synthesis

Kazunari Yosikawa^a, Naohiro Iwanaga^a, Hamachi Tatsuya^a, Akira Yamawaki^{a,*}

^aKyushu Institute of Technology, 1-1 Sensui-cho Tobata-ku Kitakyushu-shi Fukuoka-ken, 804-8550, Japan

*Corresponding Author: yama@ecs.kyutech.ac.jp

Abstract

Hardware implementation of the image processing is important for embedded systems to achieve a high performance and low power. However, the hardware design has high design load. High-level synthesis (HLS) is a technology for automatically converting high level language programs into the hardware language programs. The HLS can make the load of hardware design low. Generally we develop the image processes software using floating-point number. When converting the software using floating-point number to the hardware module, the hardware size and power consumption become large and the operation speed becomes low generally. Thus, the fixed-point software which HLS can convert is needed to acquire the efficient hardware module. The canny edge filter is one of frequently used image processes to detect the edge on the image. In addition, the canny edge filter consists of the promising primitive image processes to be used generally. Thus, we develop a canny edge filter with fixed-point number for HLS. The experiment compares hardware modules using fixed-point number and float-point number respectively. The results show that our fixed-point program can be converted by HLS to the efficient hardware module with smaller size and higher operation speed than the floating-point one.

Keywords: Canny edge filter, High-level synthesis, fixed-point.

1. Introduction

Embedded systems have to support sophisticated image processing while they have to achieve a high performance and low power consumption. Thus, the hardware implementation of image processing is very important for the embedded systems.

However, the hardware design has high design load since it requires designing a high level language expressing the algorithm, developing an HDL program expressing the behavior of the hardware and performing a clock-cycle based operational verification. Thus, the high-level synthesis (HLS) technology automatically converting software in C language into the hardware in hardware description language (HDL) has been researched and developed⁽¹⁻³⁾.

Generally we develop the image processes software using floating-point number. When converting the software using floating-point number to the hardware module, the hardware size and power consumption become large and the operation speed becomes low generally. Thus, the fixed-point software which HLS can convert is needed to acquire the efficient hardware module. Thus, we are developing the fixed-point math elementary functions with compatibility to the math library, which can be converted to the efficient hardware module⁽⁴⁾.

The canny edge filter⁽⁵⁾ is one of frequently used image processes to detect the edge on the image. In addition, the canny edge filter consists of the promising primitive image processes to be used generally. Thus, we develop a canny edge filter with fixed-point number for HLS including several generic image processes that can also be converted to each hardware module. In developing, we reuse the fixed-point math library which we have developed.

The rest of the paper is organized as follows. ...

2. Canny Edge Detector Function

2.1 Canny Edge Detector Algorithm

The canny edge filter consists four processes. First process is to remove a noise by smoothing the image. This process prevents detection of an unnecessary edge. Second

process obtains the edge intensity and the gradients' direction. Third process is a line thinning. This process uses the gradients' direction that is calculated by second process. Final process judges the edge.

2.2 Floating-point Program

First, we describe canny edge filter in floating-point. In smoothing process, we use the kernel of 5×5. This kernel is shown in Fig.1. This process performs convolution operation to the each pixel.

Second, the sobel filter obtains the edge intensity and the gradients' direction. The sobel filter performs the 3×3 convolution filter horizontally by using the kernel shown in Fig.2 (a). Then, the 3×3 convolution filter using the kernel shown in Fig.2 (b) is performed. The value generated by the former process is x and the value generated by the latter process is y. The edge intensity is calculated by Eq.1. Gradients' direction (*tan*) is able to calculate by Eq.2.

$$\sqrt{x^2 + y^2} \quad (1)$$

$$\tan = y/x \quad (2)$$

Third, the line thinning process is performed. The line thinning process uses non-maximum suppression. This process uses the gradients' direction (*tan*) that is calculated by Sobel filter. By using the *tan*, directions are categorized into 0 degree, 45 degree, 90 degree, 135 degree. Three pixels of each direction are compared. As a result, the pixel with the maximum value is left. Fig.3 shows the pseudo code of Non-maximum suppression process. The lines of 1 to 5 indicate that the direction is 0 degree. At the lines of 2 to 4, the value of pixel is 0 when the center pixel (*pixel[5]*) is not biggest value in three pixels. Similarly, the lines of 6

$\frac{1}{256}$	$\frac{4}{256}$	$\frac{6}{256}$	$\frac{4}{256}$	$\frac{1}{256}$
$\frac{4}{256}$	$\frac{16}{256}$	$\frac{24}{256}$	$\frac{16}{256}$	$\frac{4}{256}$
$\frac{6}{256}$	$\frac{24}{256}$	$\frac{36}{256}$	$\frac{24}{256}$	$\frac{6}{256}$
$\frac{4}{256}$	$\frac{16}{256}$	$\frac{24}{256}$	$\frac{16}{256}$	$\frac{4}{256}$
$\frac{1}{256}$	$\frac{4}{256}$	$\frac{6}{256}$	$\frac{4}{256}$	$\frac{1}{256}$

Fig.1. Kernel of smoothing

-1	0	1
-2	0	2
-1	0	1

(a) Horizontal

-1	-2	-1
0	0	0
1	2	1

(b) Vertical

Fig.2. Kernel of Sobel Filter

```

1 IF -0.4142 < tan <= 0.4142 THEN
2   IF pixel[5] < pixel[4] OR pixel[5] <= pixel[6] THEN
3     pixel[5] = 0
4   ENDIF
5 ENDIF
6 IF 0.4142 < tan < 2.4142 THEN
7   IF pixel[5] < pixel[1] OR pixel[5] <= pixel[9] THEN
8     pixel[5] = 0
9   ENDIF
10 ENDIF
11 IF |tan| >= 2.4142 THEN
12   IF pixel[5] < pixel[2] OR pixel[5] <= pixel[8] THEN
13     pixel[5] = 0
14   ENDIF
15 ENDIF
16 IF -2.4142 < tan <= -0.4142 THEN
17   IF pixel[5] < pixel[3] OR pixel[5] <= pixel[7] THEN
18     pixel[5] = 0
19   ENDIF
20 ENDIF

```

Fig.3. The pseudo code of Non-maximum suppression process

```

1 IF pixel[5] >= high THEN
2   pixel[5] = 255
3 ELSE IF pixel[5] <= low THEN
4   pixel[5] = 0
5 ELSE low < pixel[5] < high DO
6   IF pixel[1] >= high OR
7     pixel[2] >= high OR
8     pixel[3] >= high OR
9     pixel[4] >= high OR
10    pixel[6] >= high OR
11    pixel[7] >= high OR
12    pixel[8] >= high THEN
13     pixel[5] = 255
14   ELSE
15     pixel[5] = 0
16   ENDIF
17 ENDIF

```

Fig.4. The pseudo code of hysteresis threshold

to 10 correspond to the direction of 45 degree. The lines of 11 to 15 mean the direction is 90 degree. The lines of 16 to 20 show the direction is 135 degree.

Fourth, the process to judge the edge uses hysteresis

threshold process. Fig.4 shows a pseudo code of hysteresis threshold process. This process uses two threshold that high value and low value. If the value of pixel is bigger than high value, the pixel is edge. If the value of pixel is smaller than low value, the pixel is not edge. Moreover, if the value of pixel is high value between low value, the pixel is edge when the adjacent pixel is edge.

2.3 Fixed-point Program

We convert the floating-point canny edge filter into the fixed-point version. In smoothing process, many multiplication and division are used by performs convolution operation of Fig.1. However, the multiplication and division make hardware size larger. In contrast, the sift operation can be realized by small hardware. Therefore, the kernel is changed to use only sift operations. The approximated kernel is shown in Fig.5.

However, process image become darkened about 64%. We have to change voluntarily the value of threshold in hysteresis threshold process.

Moreover, Sobel filter process is involved square root operation. We use fixed-point square root operation library⁽⁴⁾.

3. Experiment and Consideration

We performed HLS to our fixed-point Canny edge detector program by Xilinx Vivado HLS tool to Xilinx FPGA. Used FPGA is *zynq*. Likewise, we performed HLS and implemented at FPGA for the floating-point Canny edge detector program as a compared target. Fig. 6 to 11 show the result about the number of used registers, the

$\frac{1}{256}$	$\frac{2}{256}$	$\frac{4}{256}$	$\frac{2}{256}$	$\frac{1}{256}$
$\frac{2}{256}$	$\frac{8}{256}$	$\frac{16}{256}$	$\frac{8}{256}$	$\frac{2}{256}$
$\frac{4}{256}$	$\frac{16}{256}$	$\frac{32}{256}$	$\frac{16}{256}$	$\frac{4}{256}$
$\frac{2}{256}$	$\frac{8}{256}$	$\frac{16}{256}$	$\frac{8}{256}$	$\frac{2}{256}$
$\frac{1}{256}$	$\frac{2}{256}$	$\frac{4}{256}$	$\frac{2}{256}$	$\frac{1}{256}$

Fig.5. Approximated kernel of smoothing

number of used LUTs, minimum period , maximum frequency, the number of clocks, minimum execution time.

As the result shown in Fig. 6, the number of used registers of fixed-point program decrease 76% compared with floating-point program. The result shown in Fig. 7, the number of used LUTs of fixed-point program decrease 74% compared with floating-point program. Therefore, hardware using our Canny edge filter is able to small compared with hardware using floating-point.

As the result shown in Fig. 8 to 9, fixed-point program is able to short crock period 20% compared with floating-point program. The result shown in Fig. 10, the number of used crocks of fixed-point program decrease 22% compared with floating-point program. As a result in Fig. 11, minimum execution time of fixed-point program is shorter 38% than floating-point program. Therefore, processing speed our Canny edge filter is able to fast compared with hardware using floating-point.

Moreover, Fig.12 shows two Canny edge detector images by fixed-point program and floating-point program. As you can see, there are a few differences of two images.

Fig.6. The number of registers

Fig.7. The number of LUTs

Fig.8. Minimum period

Fig.11. Minimum execution time

Fig.9. Maximum frequency

Fig.10. The number of clocks

Fig.12. Canny edge detector images

4. Conclusions

We have developed C programs the fixed-point Canny edge filter is able to be HLS. We have proven that hardware size decrease 76% as compared with floating-point program through the logic simulation and implementation experiment at FPGA. Therefore, we have been able to decrease 33% in the execution time.

As a result, we have developed highly useful hardware of fixed-point canny edge filter that is able to be converted by HLS.

As future work, we will propose efficiency index of the process that include multiple processing.

References

- (1) Wim Meeus, Kristof Van Beeck, Toon Goedemé , Jan Meel, and Dirk Stroobandt, "An overview of today's high-level synthesis tools", Design Automation for

Embedded Systems, Volume 16, Issue 3 , pp. 31-51, 2012.

- (2) Xilinx: “Xilinx Vivado Design Suite user’s guide: HLS-Xilinx”, http://japan.xilinx.com/support/documentation/sw_manuals_j/xilinx2012_2/ug902-vivado-high-level-synthesis.pdf (accessed 2th, Dec, 2013)
- (3) Impulse “Impulse C ANSI-C math library support package Xilinx “Xilinx Vivado Design Suite user’s guide: HLS-Xilinx”, http://japan.xilinx.com/support/documentation/sw_manuals/xilinx2012_2/ug902-vivado-high-level-synthesis.pdf (accessed 11th, Dec, 2013)
- (4) Kazunari Yoshikawa, Naohiro Iwanaga, and Akira Ymawaki: “Development of Fixed-point Square Root Operation for High-level Synthesis”, Proceedings of the 2nd International Conference on Industrial Application Engineering 2014, pp.16-20, 2014
- (5) John Canny: “A Computational Approach to Edge Detection”, IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. PAMI-8, NO. 6, pp.679-698, NOVEMBER 1986