Comparative evaluation between fixed and floating point hardware implementation of image conversion on high-level synthesis

Miyu Fujita[a,*], Kazunari Yosikawa[a], Naohiro Iwanaga[a], Akira Yamawaki [a,*]

[a]Kyushu Institute of Technology, 1-1 Sensui-cho Tobata-ku Kitakyushu-shi Fukuoka-ken, 804-8550, Japan

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

Abstract

High-level synthesis (HLS) automatically converts the software program in a high-level language to the hardware behavior in a hardware description language. Generally, the image processing programs have been written in floating-point format. Thus, inputting such conventional image processing programs, the HLS tool generates the hardware modules with many floating-point calculators. The floating-point calculator generally requires a large amount of hardware resources. Thus, this paper develops the fixed-point image processing programs that will be converted to the hardware module including light-weight fixed-point calculators, concentrating on the image conversions such as scaling, rotation and shear. In the experiment, we confirm the effect of the fixed-point programming for the HLS compared with the floating point one.

1. Introduction

Electronic products need more performance, multi-function and high speed for the image processing. To achieve such requirements, the hardware implementation of the image processing is mandatory for the embedded product cannot employ high performance processor like the personal computer. However, hardware development is very time-consuming and high cost work. Thus, the HLS have been researched and developed to reduce the burden due to hardware development.

For the software development, many useful image processing library exist like OpenCV. Generally, the image processing programs have been written in floating-point format. Thus, inputting such conventional image processing programs, the HLS tool generates the hardware modules with many floating-point calculators. The floating-point calculator generally requires a large amount of hardware resources. Thus, the fixed-point image processing programs that an HLS tool can convert to the hardware modules including light-weight fixed-point calculators are needed.

We have developed a synthesizable math library with fixed-point elementary functions such as the trigonometric functions, the exponential function and the logarithmic function. However, more sophisticated process like image filter including them have not been developed.

This paper develops the fixed-point image processing programs, concentrating on the image conversions such as scaling, rotation, shear and interpolation. In the experiment, we confirm the effect of the fixed-point programming for the HLS compared with the floating point one.

The rest of the paper is organized as follows. Section 2 explains the overviews of targeted image conversions. Section 3 explains method of convert programs from floating-point type to fixed-point type. Section 4 shows the circuit scale and processing time comparison result of the fixed-point type and floating-point type. Section 5 describes the conclusion.

2. Image conversion

2.1 Affine transformation

The affine transformation is a linear transformation between the vector space. In this study, we have created the respective conversion programs by using the affine transformation. Eq.1 is the basic equation of affine transformation. The input coordinates are (x, y) and the output coordinates are (x ', y')

$$[\,x' \quad y' \quad 1] = [\,x \quad y \quad 1] \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{bmatrix} \qquad (1)$$

DOI: 10.12792/iciae2015.108

2.2 Parallel shift

As Eq.2, t_x is the intercept of the x-axis, and t_y is the intercept of the y-axis. Therefore, when the value of t_x and t_y is changed, image is moved in each x direction and y direction.

$$[\ x' \quad y'\ 1\ ]=[\ x \quad y\ 1\ ]\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ t_x & t_y & 1\end{bmatrix} \qquad (2)$$

2.3 Scaling

As Eq.3, the coefficient a is the magnification of the image size to the x direction, and the coefficient d is the magnification of the image size to the y direction.

$$[\ x' \quad y'\ 1\ ]=[\ x \quad y\ 1\ ]\begin{bmatrix}a & 0 & 0\\ 0 & d & 0\\ 0 & 0 & 1\end{bmatrix} \qquad (3)$$

2.4 Rotation

For the coefficients of a, b, c and d in Eq.1, the values of each trigonometric are set as Eq.4. By doing so, as a reference coordinates (0, 0), the image is rotated by any angle θ degrees. For the coefficients of t_x and t_y in Eq.1, the coordinates of the center of the image (c_x,c_y) are set. By doing so, the image is rotated with respect to (c_x,c_y).

$$[\ x' \quad y' \quad 1]=[\ x \quad y \quad 1]\begin{bmatrix}cos\theta & sin\theta & 0\\ -sin\theta & cos\theta & 0\\ c_x & c_y & 1\end{bmatrix} \qquad (4)$$

2.5 Shear

In the equation (1), b is the coefficient of y in the equation of x ', and c is the coefficient of x in the equation of y '. By b and c have values, image is tilted as shown in the Fig1.

$$[\ x' \quad y' \quad 1]=[\ x \quad y \quad 1]\begin{bmatrix}0 & b & 0\\ c & 0 & 0\\ 0 & 0 & 1\end{bmatrix} \qquad (5)$$

Figure 2 shows the images after each conversion. As shown in Figure 2, we have succeeded in converting images exactly by using Affine transformation.

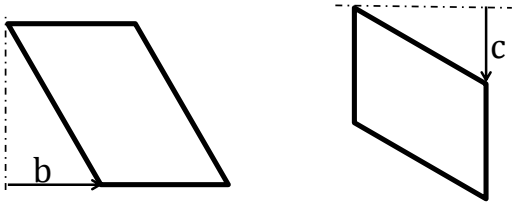

Fig1. Shear

(a) Original image

(b) Scaling

(c) Rotation

(d) Shear

Fig2. Images after conversion

2.6 Images interpolation

In this study, we use the Bilinear Method as a method of interpolation. Bilinear method is interpolation method that put the average of the color values of the target pixel and eight surrounding pixels surrounding to target pixel. It is too simple method compared to other methods. Large distortion is not seen in the image after conversion by each conversion program that was created in this study, and this study is focused on the point that is speeding up. So we used this

interpolation technique in this study.

3. Fixed-point Image Conversion

Scale and shear programs require the values for determining the magnification of images size and tilt condition of images. These values are often used as a decimal.

Therefore, we bet the power of 2 of N of their value in the first (N-bit left shift). N is the number of bits of the integer part of the fixed-point that was decided arbitrarily on the program. By doing so, the fractional value is treated as integer type by temporarily large value. Then, the value can hold the information of the fractional part. When using a final calculation result as an output, the calculation result is divided by 2 of n squared (n-bit right shift). By doing so, the value temporarily increases is returned to the original output value.

In the rotation, method is the same as the scale and shear basically. At the time of conversion, value decided by user is the angle only, but, in this study, the rotational angle is being limited to an integer. So, we were pre-declare the value that the circumference ratio was n-bit left shift, and used its value as circumference ratioπ. The trigonometric functions were calculated by the existing fixed-point type trigonometric calculation program.

4. Comparison with floating-point type program

4.1 Hardware Parameters

We used an HLS of Vivado HLS 2014.2. The sizes of the generated hardware of fixed-point and floating-point version are evaluated. Table 1 shows these comparison results.

Table 1 represents that the number of look-up tables (LUTs), flip-flops, and multipliers used. The fixed-point programs have decreased hardware resources significantly compared to floating-point programs. In other words, it is indicates that the circuit scale is significantly reduced.

4.2 Performance Evaluation

We actually simulated conversion programs created in this study by Vivado 2014.2, and measured processing time and confirmed the image after HLS. We will explain about the simulation method using the image. First, outputting the image data to a text file as text data in binary. Second, loading the text data to the test bench by using the interface called TEXIO. Finally moving the program by using the loaded image data as input, and observing simulation waveform. Table 2 shows a processing time [ns] of each conversion program at 1pixel obtained from the simulation waveform.

The experimental results indicate that the processing time of the fixed-point type is shorter compared to the floating-point type. And, error rate of "data of the image after each conversion by c language program" and "data of the image after each conversion by VHDL after HLS" became 0. In other words, images after conversion were confirmed that it is no problem.

Table1. Circuit scale of each conversion program

	scaling		rotation	
	floating	fixed	floating	fixed
SLICE	447	53	2870	283
LUT	1215	69	7772	720
FF	825	196	6392	788
multipliers	7	1	106	9
Embedded memory	0	0	0	0
Shifter	0	0	0	0
Clock minimum period	8.059	3.207	9.413	5.571
	shear		parallel shift	
	floating	fixed		
SLICE	571	122	55	
LUT	1534	172	95	
FF	1130	394	194	
multipliers	17	13	1	
Embedded memory	0	0	0	
Shifter	0	36	0	
Clock minimum period	7.805	4.062	3.425	

Table2. Processing times in 1pixel of each conversion program [ns]

	floating	fixed
scaling	100	20
rotation	180	40
shear	140	20

5. Conclusion

We have developed fixed-point programs that can be converted to the hardware modules by HLS tool. Compared with the floating point versions, our programs were able to

significantly reduce the circuit scale. In each of the functions, the number of SLICE has been reduced by about 80%, the number of LUT has been reduced by about 90%, and the number of flip-flop has been reduced by about 70%. Number of multipliers has reduced about 90% in function of the scaling and rotation, has reduced about 25% in function of the shear. In addition, the processing speed is also much faster, processing speed in each of the conversion program was about 5 times. As a result, we have succeeded in improving the processing speed and reduce the circuit size. This fact promises an object of this study.

References

(1) Wim Meeus, Kristof Van Beeck, Toon Goedemé , Jan Meel, 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: HLSXilinx” 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: HLSXilinx” http://japan.xilinx.com/support/documentation/sw manuals/xilinx2012 2/ug902-vivado-high-level-synthesis.pdf (accessed 11th, Dec, 2013)