Development of Fixed-point Trigonometric Function Library for High-level Synthesis

Naohiro Iwanaga*, Takayoshi Abe, Akira Yamawaki

Kyushu Institute of Technology, Fukuoka, Japan(804-8550)

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

Abstract

The sophistication of embedded devices needs to make various processes like image processing, audio processing and data searching and so on as hardware to achieve high-performance and low-power dissipation. To reduce the burden of hardware design, high level synthesis (HLS) converting an algorithmic description such as C program to the hardware description language has been paid attention. In HLS flow, several libraries including elementary functions such as trigonometric and exponential functions in the algorithm are provided. However, they are implemented by using floating-point number, which is not suitable for hardware implementation in view of the hardware size and operation speed. Therefore, this paper attempts to develop the fixed point trigonometric functions (sine and cosine) by using CORDIC. Compared with a conventional floating point HLS library based on math.h, we demonstrate that our fixed point library has almost same accuracy as it while making the hardware size small.

Keywords: High-level-synthesis, CORDIC, VHDL.

1. Introduction

Recently, as embedded devices require more high performance and miniaturization, hardware implementation of sophisticated processing like the moving image processing is needed. In many cases, such processing includes the calculations of elementary functions such as trigonometric functions, exponential functions and so on.

Typically, at first, the developers make the algorithm in a programming language such as C language. Then, he/she ports it to hardware design in the hardware description language (HDL). To reduce the burden of hardware design, high level synthesis (HLS) converting an algorithmic description such as C program to the hardware

description language has been paid attention. By using HLS tool, the developers can realize the hardware module on the unified design flow from C programming, design verification to hardware implementation. However, when using the elementary functions in the design flow, several problems arise.

Traditionally, the elementary functions have been provided as intellectual property (IP) of HDL program or netlist⁽¹⁾. In this case, the simulation time of C program expands significantly because the logic simulator is invoked to simulate HDL program or netlist. Also, for the provided IP, parameters may not be changed easily. The device dependency of IP may lead to a difficult problem. That is, the advantage of using C program is eliminated.

Some HLS tools provide C libraries including elementary functions⁽²⁻³⁾. In such tools, the developer can make hardware module automatically without suffering from the logic simulator and device dependency. However, they are implemented by using floating-point number, which is not suitable for hardware implementation in view of the hardware size and operation speed.

This paper attempts to develop the fixed point trigonometric functions (sine and cosine) by using CORDIC algorithm⁽⁴⁻⁷⁾. Since our library contains only fixed-point arithmetic units, it may improve a hardware performance and circuit size. Compared with a conventional floating point HLS library based on math.h, we demonstrate that our fixed point library has almost same accuracy as it while making the hardware size small on FPGAs.

2. Software of Trigonometric Function

2.1 Trigonometric Function by CORDIC

In this study, in order to produce a superior hardware in view of the hardware size and operation speed, using the

algorithm called (COordinate Rotation Digital Computer) CORDIC. The CORDIC is an algorithm that represents approximately the elementary function arithmetic. In this case, function is the vector in two-dimensional surface. And the solution of function is able to be solved rotating the vector. Formula (1) to (3) is a formula for solving the trigonometric functions in CORDIC algorithm.

$$x[j + 1] = x[j] - \sigma_j 2^{-j} y[j] \quad (1)$$

$$y[j + 1] = y[j] + \sigma_j 2^{-j} x[j] \quad (2)$$

$$z[j + 1] = z[j] - \sigma_j \tan^{-1}(2^{-j}) \quad (3)$$

σ_j is the sign representing the direction of rotating the vector, j is the number of repeat count, x and y is the coordinates. Then, $\tan^{-1}(2^{-j})$ was used in formula (3) and the number of repeat count is about 17 times. Therefore $\tan^{-1}(2^{-j})$ is used as a constant. Next, using a right triangle with the angle shown in Table 2.1, it will be described solving trigonometric graphically. In this case, initial angle is 30 degrees and P_0 is set as first vector. First, hypotenuse of a right triangle when $j = 0$ is superposed with the base of the right triangle when $j = 1$. Then, in order to determine the orientation superimposing, to perform size comparisons angles. Because 45 degrees ($j=0$) is higher than 30 degrees (input value), right triangle is superimposed as shown in Fig2.1. And P_1 is set as vector after superposition. Since superimposed on the negative direction, the size comparison of the angle of the next, it is assumed to use 18.435 to minus 26.565 to 45. Similarly, hypotenuse of a right triangle when $j = 1$ is superposed with the base of the right triangle when $j = 2$. Because 18.435 degrees ($j=1$) is lower than 30 degrees (input value), right triangle is superimposed as shown in Fig2.2. And P_2 is set as vector after superposition. Since superimposed on the positive direction, the size comparison of the angle of the next, it is assumed to use 32.471 to 14.036 plus to 18.435.

Thus, by superposing by the angle comparison, the right triangle to $j = 17$ from $j = 0$, it is possible to approximate the input 30 [deg] the angle of the vector P_{17} finally. X-coordinate is 1.4261305 and y-coordinate is 0.82339 of the final vector. Divided by $r = 1.64760258$ vector absolute values x coordinate and y coordinate final for calculating the cosine and sine. Thus, it can be approximately calculated $\sin(30)$ is 0.49975, $\cos(30)$ is .865579. Further, it is possible to use as a constant r is the same value even after any rotation in any input.

Fig.1. State of convergence in the case of $Z = 30[\text{deg}]$

Fig.2. State of convergence in the case of $Z = 30[\text{deg}]$

2.2 Change in Fixed-point

In this study, formula of the CORDIC algorithm is realized by only fixed-point arithmetic, because the purpose of this study is implementation by high-level synthesis. Range of input (Z) is from 0 to 2π and the output is cosine and sine that is a fixed point format 30-bit integer part 2 bit fraction part.

Flow chart in Fig2.3 is the flow of the processing of the trigonometric functions of the software in fixed-point arithmetic. Further, in order to simplify the description, the flowchart is assumed that output the results of only the first quadrant. For each value, Z is the input angle. The value that shifted Z to the 16 bits left is $Z[1]$. And $Z[0]$ is the angle to be used for comparison with $Z[1]$. Then, we compared the size Z of the $[1] Z[0]$, is performed to correct the angular movement and the coordinates according to the size. Processing by the comparison is repeated until $j = 16$ from $j = 0$, the final coordinates are calculated.

Fig.3. Fixed-point version of the flow chart

When formula of the algorithm without any changes is described by the program, the performance of the hardware is lowered in view of the hardware size and operation speed. Therefore, we improved the two for performance improvement. First, division of powers of 2 is included in the formula of the CORDIC algorithm to correct the coordinates. In this study, we adopted the bit shift to the right, not the division. Furthermore, division of r is included in processing. In this study, because r is constant, instead of dividing the final coordinates, we have achieved the same operation by dividing the first coordinates. Thanks to this improvement, it is possible to prevent the formation of the divider and to generate hardware better in view of the hardware size and operation speed.

3. Experiment

3.1 Experiment environment

In order to perform logic synthesis, we used the ISE of Xilinx Inc. And in order to perform the high-level synthesis, we used vivado HLS of Xilinx Inc. In this study, the processs was selected sequential processing and the

pipelined processing by vivado HLS. Math.h standard library of C language were prepared to compare the fixed-point and floating-point in view of the hardware size and operation speed.

3.2 Mounting result

To examine the resource utilization of implementing the Virtex4, Virtex5, Virtex6 each VHDL the selected program VHDL the selected program the type sequential processing (my_sin_cos_seq), the pipelined processing (my_sin_cos_pipe).The resource utilization on each device was shown in Table 1 and Table 2. It was found that the results of these and can be used in FPGA various processing any type, such as pipeline type sequential, can be used to process the trigonometric functions than the majority of the FPGA. Next, Table 3 shows the utilization of the resources of implementing the math_sin_cos that the high-level synthesis and cosf and sinf of math.h. Resource utilization are shown in Table 3, is larger than the resource utilization are shown in Table 1 was selected type serial processing, in that it allows reducing the resource, it can be seen that more of the fixed-point arithmetic are appropriate for the hardware. And Table 4 shows the results of evaluation of the error in the output of my_sin_cos_seq for the output math_sin_cos.

Table 1. Resource utilization of my_sin_cos_seq

FPGA	Resource	Used	Available	Utilization[%]
Virtex4	Registers	244	10944	2
	LUTs	1045	10944	10
Virtex5	Registers	246	12480	2
	LUTs	751	12480	6
Virtex6	Registers	238	93120	1
	LUTs	449	46560	1

Table 2. Resource utilization of my_sin_cos_pipe

FPGA	Resource	Used	Available	Utilization[%]
Virtex4	Registers	1103	10944	10
	LUTs	3570	10944	35
Virtex5	Registers	1103	12480	10
	LUTs	3323	12480	27
Virtex6	Registers	1100	93120	1
	LUTs	1664	46560	3

Table 3. Resource utilization of math_sin_cos

FPGA	Resource	Used	Available	Utilization[%]
Virtex4	Registers	1992	10944	18
	LUTs	6742	10944	61
Virtex5	Registers	1970	12480	15
	LUTs	3306	12480	26
Virtex6	Registers	1907	93120	2
	LUTs	3102	46560	6

Table 4. Error

Z[rad]	Error_sin	Error_cos
0	-	0.176%
0.1745	0.387%	0.174%
0.3491	0.054%	0.168%
0.5236	0.056%	0.158%
0.6981	0.110%	0.145%
0.8727	0.145%	0.110%
1.0472	0.158%	0.056%
1.2217	0.168%	0.054%
1.3963	0.174%	0.387%
1.5708	0.176%	-

The result as shown in Table 4, the error is 0.1 to about 0.4%. From the results, the hardware of this study became perform a reduction in resources by fixed-point arithmetic and error became a practical range.

4. Conclusions

We have developed C programs of sine and cosine with completely fixed point number that can be converted by high-level synthesis. Compared with a conventional floating point HLS library based on math.h, we demonstrate that our fixed point library has almost same accuracy as it while making the hardware size small.

As future work, we will design other elementary functions as fixed point C program by using CORDIC.

References

- (1) Xilinx: "LogiCORE IP CORDIC v4.0", Xilinx, DS249, 2011.
- (2) Xilinx: "Vivado HSL", <http://www.xilinx.com/>, 2013.
- (3) Impulse Accelerated Technologies: "Impulse CoDeveloper", <http://www.impulseaccelerated.com/>,

2013.

- (4) Milo's D. Ercegovac and Tom'as Lang : "DIGITAL ARITHMETIC", Elsevier Science, Volume 11, 2003
- (5) B.Lakshmi and A.S.Dhar : "CORDIC Architectures: A surevey", Hindawi Publishing Corporation, Volume 2010, ArticleID 794891, 19pages
- (6) J. S. Walther , "A unified algorithm for elementary functions", Hewlett-Packard Company, Palo Alto, California
- (7) Jack E. Volder : "The CORDIC Trigonometric Computing Technique", IRE Trans Electron Comput, 1959