Development of Fixed-point Square Root Operation for High-level Synthesis

Kazunari Yoshikawa^{a*}, Naohiro Iwanaga^a, Akira Yamawaki^a

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

*Corresponding Author: yoshikawa@boss.ecs.kyutech.ac.jp

Abstract

High-level synthesis (HLS) is a technology for automatically converting software language programs into the hardware description language (HDL) programs. When performing HLS to a software program that includes elementary functions of floating point number, hardware size and operation speed of the converted hardware increase. We develop a fixed-point generic HLS library of elementary functions that is able to be converted by HLS. This paper focuses on square root operation in elementary functions. We show how to design a fixed-point program in C language of the square root by the coordinate rotation digital computer, CORDIC algorithm. Then, we convert it to the HDL program by Xilinx Vivado HLS tool. The experiment compares our hardware generated with the conventional floating point hardware generated from the math library. The results show that our fixed-point square root can decrease hardware size and operation speed than the conventional one. In addition, the accuracy of our fixed-point square root is almost same as the floating point square root generated from the math library; the error is less than 0.2%.

Keywords: High-level synthesis, fixed-point, square root, CORDIC

1. Introduction

In order to realize high-performance embedded device with low-power consumption, the computational processes must be realized by the hardware instead of the software. However, the hardware design is a heavy task badly affecting the development period and cost. 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].

In general, signal processing such as audio processing, image processing uses elementary functions such as square root, trigonometric functions, logarithm, and so on. Conventionally, HLS tool supports elementary functions with compatibility to the math library of C language. Thus, they employ floating-point number. However, converting hardware of floating-point function has problem from the viewpoint of the processing time, the hardware size and the power consumption.

In contrast, we are developing a generic library including the fixed-point elementary functions that has same functions as the math library in C language and can be converted to the hardware by any HLS tool. This paper focuses on square root operation in elementary functions.

To develop fixed-point square root, we employ coordinate rotation digital computer, CORDIC algorithm. Operations in CORDIC are primarily additions and shift operations. They are suitable for hardware. In addition, CORDIC is able to realize various elementary functions such as trigonometric functions, logarithm by changing some parameters and operations. This feature indicates that an increase of hardware size is able to be suppressed for sharing of CORDIC modules, when converting to hardware operations that use plural elementary functions.

However, the range of input value is narrow by using CORDIC correctly. This paper shows a pre-process mapping an input value into the appropriate range, a main part of fixed-point CORDIC algorithm and a post-process expanding to the original range.

In the experiment, following are performed, being compared with *sqrtf* function of math library.

(1) The appropriate range of input value for CORDIC

is revealed.

(2) CORDIC approximately obtains results by repeating the main loop. The number of repetitions increases the processing time. While, the smaller number of repetitions leads to the higher error. Their optimal parameters are investigated by making trade-off among the number of repetitions, the errors and the processing time.

(3) Effectiveness of our square root operations is shown by being compared with *sqr* function about the error, the processing time and the amount of hardware.

To the best of the author's knowledge, nobody has performed such experiments.

The rest of the paper is organized as follows. Section 2 describes the original CORDIC algorithm of square root operation including pre-process and post-process. Section 3 shows algorithm of the fixed-point square root operations. Section 4 shows experimental results and discusses about their result. Finally, Section 5 concludes this paper.

2. Square Root Operations by CORDIC

First, we describe the original square root by CORDIC with floating-point. We think that the input value of the CORDIC square root is expressed by the form as shown in Eq. 1. The a is the input value. The $a * k^2$ is within the appropriate range for the CORDIC. We set k to 2. This is because the power of 2 can be realized by shift operation. This feature is desired in fixed-point.

$$k\sqrt{a} = \sqrt{a * k^2} \quad (1)$$

The range where CORDIC can correctly execute has a lower bound and an upper bound. When the input value is less than the lower bound, the pre-process multiplies the input value by 4 until its result enters within the range. Otherwise, the pre-process divides the input value by 4 as well. After the CORDIC square root, the post-process returns the calculated value into the original range by multiplication of 2 or division of 2.

Fig. 1 shows a pseudo code of square root operations programs by CORDIC.

At the lines of 2 to 8, the pre-process reforms the input value to give the CORDIC the appropriate value. The s is a lower bound and the t is an upper bound of the appropriate range for the CORDIC.

At the lines of 2 to 4, the small input value is multiplied by 4 until the multiplied value becomes bigger than the s . Moreover, the number of multiplications is recorded into

the m . Likewise, at the lines of 5 to 8, the large input value is divided by 4 until it becomes smaller than the t . Moreover, the number of divisions is recorded into the n .

Then, we perform square root operations by CORDIC at the lines of 9 to 20. CORDIC has two modes that are rotation mode and vector mode. Changing their modes is able to compute various functions. Square root operations are used vector mode. Square root operations of vector mode are expressed by Eq. 1 to Eq. 7.

$$x[0] = a + 1 \quad (2)$$

$$y[0] = a - 1 \quad (3)$$

$$x[j+1] = x[j] + \sigma_j 2^{-j} y[j] \quad (4)$$

$$y[j+1] = y[j] + \sigma_j 2^{-j} x[j] \quad (5)$$

$$\sigma_j = \begin{cases} 1, & y[j] < 0 \\ -1, & y[j] \geq 0 \end{cases} \quad (6)$$

$$z = x / 1.65632 \quad (7)$$

x and y are coordinates. j is the number of repetitions. σ_j is direction of vector rotation. z is output value.

Then, at lines of 21 to 22, the post-process returns z within the original range, performing some multiplications or divisions according to the recorded number in the m and n .

```

1      Input      a
2      WHILE a < s DO
3          a = a * 4
4          m = m + 1
5      WHILE a > t DO
6          a = a / 4
7          n = n + 1
8      ENDWHILE
9      x = a + 1
10     y = a - 1
11     FOR i = 1 TO j DO
12         IF y < 0 THEN
13             x = x + y / 2i
14             y = y + x / 2i
15         ELSE y >= 0 DO
16             x = x - y / 2i
17             y = y - x / 2i
18         ENDIF
19     ENDFOR
20     z = x / 1.65632
21     z = z * 2n-m
22     Output      z

```

Fig. 1 The pseudo code in floating-point.

3. Square Root Operations in Fixed-Point

We convert the floating-point square root operations into the fixed-point square root operations. The allotment of bits in fixed-point is that an integer part is p bits, a fractional part is q bits, number of bits shift in computation on the way is r bits. Therefore, a must be performed q bits left shift. Fig. 2 shows a pseudo code of square root operations program in fixed-point.

```

1      Input      a
2      WHILE a < s DO
3          a = a << 2
4          m = m + 1
5      WHILE a > t DO
6          a = a >> 2
7          n = n + 1
8      ENDWHILE
9      x = a + (1 << q)
10     y = a - (1 << q)
11     FOR i = 1 TO j DO
12         IF y < 0 THEN
13             x = x + (y << i)
14             y = y + (x << i)
15         ELSE y >= 0 DO
16             x = x - y / (y << i)
17             y = y - x / (x << i)
18         ENDIF
19     ENDFOR
20     z = x * u
21     z << (n - m - r)
22     Output      z

```

Fig. 2 The pseudo code in fixed-point.

At the lines of 2 to 8, a power of 2 is shift operations in fixed-point. Therefore, the pre-process is that multiplication of 4 is 2 bits left shift and division of 4 is 2 bits right shift. At the lines 9 to 20, they are convert CORDIC of square root operations in floating-point into that in fixed-point. At the lines of 9 to 10, 1 of addition is performed q bits left shift. At the lines of 11 to 19, division of 2 is 1 bit right shift. At the line of 20, u is value of $(1/1.65632 \ll r)$ in fixed-point.

The post-process returns z within the original range, performing some left or right shift according to the recorded number in the m and n . However, z is performed r bits left shift in line of 20. Therefore, the post-process must perform r bits right shift. Accordingly, at the line of 21, the post-process perform to return z within the original range

and shift number of bits shift in computation on the way.

According to the above processes, square root of a is able to computed for z with fixed-point.

4. Experiment and Consideration

4.1 Determination of input value range of CORDIC

First, we regard that whole fixed-point format of square root operations is 32 bits. The format of 32 bit is 12 bit of the integer part and 20 bit of the fractional part. Therefore, according to suppose that input data is the stream data A/D converted output from the sensor. In other words, the integer part is 12 bit because resolution of general A/D convertors is 8-12 bit. Accordingly, input value range of square root operations are 0.0 to 4095.9999990463.

Then, we perform square root operations range of 1.0-100.0 that don't perform the pre-process and the post-process for determination of input value range for CORDIC. We performed the error comparison with *sqrtf* function of math library. As a result, the error is less than 0.2% in range 2.25 to 9.0. This paper adopts this range.

4.2 Determination of the number of repetitions

At the repetitive operation for CORDIC by Eq. 4 to Eq. 6, we searched the maximum error of our function that compare math library (m-lib), the number of repetitions increase one by one from two determine for the appropriate number of repetitions. Fig. 3 shows this result in.

Fig. 3 Maximum error for the number of repetitions.

The result as shown in fig.3, the maximum error saturated that the number of repetitions is five or more times. Also, the maximum error of five is 0.151466 (%) that is the lowest. This result was a practical error. Therefore, we consider only the case of five or more times the number

of repetitions in later.

4.3 Consideration of the number of clocks

We performed the logic simulation in 5 to 13 times in the number of repetitions to ascertain the number of clocks. Fig. 4 shows this result and *sqrtrf* function result of math library.

Fig. 4 Compare of the number of clocks.

As the result shown in fig. 4, as the number of clocks decreases the number of repetitions decreases. Also, the minimum number of clocks is 12 when the number of repetitions is five. This is bigger than 10 that the number of clocks of math library. However, its difference is few.

4.4 Implementation experiment at FPGA

We performed HLS to our fixed point square root with the number of repetitions from 5 to 13 by Xilinx Vivado HLS tool to Xilinx FPGA. Used FPGA is *zynq*. Likewise, we performed HLS and implemented at FPGA for the floating point *sqrtrf* of the math library as a compared target.

Fig. 5 to 10 show the result about registers used, LUTs used, minimum period, maximum frequency, minimum execution time.

Fig. 5 Comparison of registers used.

Fig. 6 Comparison of LUTs used.

Fig. 7 Comparison pf minimum period.

Fig. 8 Comparison of maximum frequency.

Fig. 9 Comparison of minimum execution time.

As the result shown in Fig. 5, registers used of our operations decrease 29% compared with math library. The result of Fig. 6 indicates that LUTs used of our operations is almost same as the math library. Therefore, our hardware can decrease the hardware size compared with math library.

As the result shown in Fig. 7 to 8, our hardware improves the clock period and clock frequency of 26% than the math library. Therefore, as the result shown in fig. 9, minimum execution time of our operations is shorter than one of math library, when the number of repetitions is 5 to 7. Accordingly, we are able to improvement in processing speed.

5. Conclusions

We have developed C program of fixed-point square root operations by using CORDIC that is able to be converted by HLS tool to the hardware module automatically.

Through some experiments, several facts that have never been investigated have been revealed. The range of input value is narrow by using CORDIC. According to the basic experiment, we have proven that the error with *sqrtrf* function of math library is less than 0.2% when input value of CORDIC is range of 2.25 to 9.0. Therefore, we have made an addition the pre-process mapping an input value into the appropriate range of CORDIC and the post-process expanding to the original range.

Moreover, CORDIC approximately obtains results by repeating the main loop. When this number of repetitions is five or more times, we have proven that the above error is kept.

We have proven that hardware size decrease 29% as compared with math library through the logic simulation and implementation experiment at FPGA. Therefore, we have been able to decrease 26% in the execution time. According to the above processes, we have developed highly useful hardware of fixed-point square root operations that is able to be converted by HLS

As future work, we will design other elementary functions included math library as fixed-point that is able to be converted by HLS.

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)

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,