Optimized Hardware Algorithm for Integer Cube Root Calculation and Its Efficient Architecture

Rachmad Vidya Wicaksana Putra[1], Trio Adiono[2]
Microelectronics Center
Institut Teknologi Bandung, Indonesia
[1] rachmad@pme.itb.ac.id | rachavidyawp@gmail.com, [2] tadiono@stei.itb.ac.id

***Abstract*—Scientific applications, digital signal processing, and multimedia usually need to compute a large number of arithmetic operations. One of them is cube root operation. It is one of the fundamental arithmetic operation which is not received much attention. Because of its calculation complexity, cube root is difficult to implement in Field Programmable Gate Array (FPGA). Hence in this paper, we propose an optimized hardware algorithm for integer cube root calculation and its efficient architecture. Integer cube root calculation is computed by using 3-digits of binary number and iterative calculation. An optimized hardware algorithm idea is reducing computational complexity in factor generator unit. For design evaluations, we use 32-bit integer cube root architecture and simulate it with several test vectors. Evaluation results show us that the design architecture is valid. The design latency is defined by (*N*/3)+2, with *N* is bit-width of the design input. Hence, 32-bit design will be executed only in ((32+1)/3)+2 = 13 clock cycles. The design also has been synthesized for several FPGA implementation with promising results in area consumption and speed.**

Keywords—Cube root; optimized hardware algorithm; FPGA; efficient architecture.

I. INTRODUCTION

Scientific applications, digital signal processing, multimedia, and 3D graphics applications usually need to compute a large number of arithmetic operations, such as square root, cube root, logarithm, trigonometric functions, and etc [1], [2]. Cube root operation is one of the fundamental arithmetic operation which is used in many applications but not received much attention [2]. There are only few proposals about cube root computation, especially about its implementation in Field Programmable Gate Array (FPGA) [1]. Because of its calculation complexity, cube root is difficult to implement in FPGA. Hence, hardware algorithm and VLSI architecture studies on cube root calculation will give opportunities to explore and implement cube root in FPGA efficiently.

Several publications about cube root algorithm and its FPGA implementation have been presented. An algorithm of cube root computation for integer operand was proposed in [3]. It discussed the integer extraction for performing cube root operation. Furthermore, there were also some publications that discussed about the arithmetic explorations. For example, the *k*-th root calculation is performed by extending the formula from square root and cube root algorithms as discussed in [4]-[6]. Besides arithmetic explorations, There were also publications that discussed about hardware implementations. Paper [1] proposed a FPGA implementation of a binary32 floating point cube root which comply with IEEE 754-2008 standard. This design uses Newton-Raphson method for data execution. It consumes 230 slices and 12 Dsp48s area, and 19 clock cycles latency. Another design is radix-2 cube root architecture [2]. It was estimated to consume 7135 *nand2* standard cell and 35.1 τ_{INV} delay of critical path (τ_{INV} is delay of an inverter with four fan-out FO4).

From literature studies, we conclude that the exploration of cube root calculation were mostly on arithmetic algorithm. Only few papers implemented the designs in FPGA. Hence, we have a target to design a low-complexity architecture of integer cube root calculation and implement it in FPGA efficiently. Our design methodology is started from mathematical approach, a cube formula. We do a reverse calculation to create a cube root formula. From the obtained formula, we optimize the calculation in order to make it suitable with hardware implementation in FPGA. Hopefully, this research can contribute in exploration of cube root algorithm and its FPGA implementation studies.

This paper is organized in the number of sections. Section I is introduction about the research backgrounds and brief descriptions of the several past researches. Section II is a brief explanation of the related theories. Section III is explanation of the proposed algorithm. Section IV is the proposed architecture. Section V is the design evaluation test and analysis. Last main section is a concluding remark. This paper is enclosed with acknowledgment and references respectively.

II. RELATED THEORIES

The integer cube root algorithm is basically derived from simple equation. We know that cube root function is inverse function from cube function. For each integer number x, we can represent it as two digits number p and q as mentioned in [7]. According to [7], we can calculate a cube operation from $x = pq$ by using standard cube formulation but with special addition operations. Hence, if we extend (1) with cube operation, we will obtain equations (2) and (3).

$$x = pq \; ; \quad p \text{ and } q \text{ are digits of decimal number} \tag{1}$$

$$x^3 = (pq)^3 \tag{2}$$

978-1-4673-6499-7/15/$31.00 ©2015 IEEE

$$x^3 = p^3 \,(+)\, 3p^2q \,(+)\, 3pq^2 \,(+)\, q^3 \quad (3)$$

Operator $(+)$ is not standard addition, but special addition. For easier understanding, we will demonstrate the equation with exact integer number. If we take $x = 56$ (decimal), then $p = 5$ and $q = 6$, we can compute (3) with substitution procedure and we obtain (4) and (5). In order to process equation (5), we need to arrange the addition calculations of 125 $(+)$ 450 $(+)$ 540 $(+)$ 216 become ladder-like addition process as shown in Fig. 1.

$$x^3 = 5^3 \,(+)\, 3 \bullet 5^2 \bullet 6 \,(+)\, 3 \bullet 5 \bullet 6^2 \,(+)\, 6^3 \quad (4)$$

$$x^3 = 125 \,(+)\, 450 \,(+)\, 540 \,(+)\, 216 \quad (5)$$

$x^3 = p^3$: 125
$3p^2q$: 450
$3pq^2$: 540
q^3 : 216
+
175616

Fig. 1. Ladder-like addition process

From the calculation processes, we can see that $(+)$ means addition with specific order of digit position. Hence, we have to be aware when doing the calculation. By using this approach, we can implement the same calculation process to binary numbers. This structure is fundamental architecture for cube operation, thus we do not need any major modifications.

III. Proposed Algorithm

A. Generic Algorithm

Actually, the proposed algorithm is inspired by (1) - (3). But, the equation is modified based on binary form in order to make the design suitable for FPGA implementation. Assume we take (1) as initial equation to modify, we will get initial equations as (6) - (8).

$$x = ab \;; \quad a \text{ and } b \text{ are digits of binary number} \quad (6)$$

$$x^3 = (ab)^3 \quad (7)$$

$$x^3 = a^3 \,(+)\, 3a^2b \,(+)\, 3ab^2 \,(+)\, b^3 \quad (8)$$

We can see that (8) consists of two main parts, independent and dependent segments. Independent segment consists of a^3 and b^3, while dependent segment consists of $3a^2b \,(+)\, 3ab^2 \,(+)\, b^3$. Independent segment means there is only one variable involved, meanwhile dependent segment means there are two or more variables involved.

If we compute the independent segment, result from a^3 will contribute dominant in the most significat number. In binary term, we call it the most significant bit (MSB). Meanwhile, result from b^3 will contribute dominant in the least significant number. In binary term, we call it the least significant bit (LSB). In this approach, b^3 will not be treated as independent segment, but treated as part of dependent segment. It is because b^3 will also contribute in dependent segment. Its dependent behavior is shown in Fig. 2. The dependent segment ($3a^2b \,(+)\, 3ab^2 \,(+)\, b^3$) will be computed just like iterative calculation as many as calculations needed between MSB and LSB. In order to understand the process easily, Fig. 2 shows us the calculation process in the cube root step-by-step.

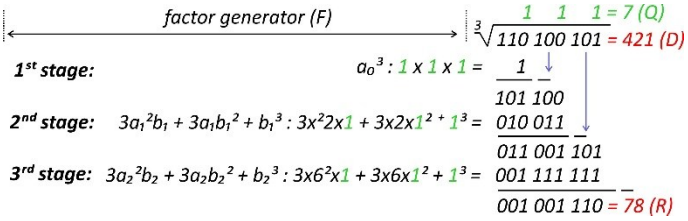

Fig. 2. Step-by-step cube root calculation

If we observe the cube root calculation step-by-step in Fig. 2, we can see that there are several pattern involved. It is slightly similar with square root architecture in our previous works [8], [9]. Firstly, the input data D is divided into several 3-digits subgroups. If the original data D bit-width can not be exactly divided by 3, it has to be appended with zero "0" as MSB instead. Secondly, the first calculation is derived to eliminate the most significant value from data D. It is predicted to be the result of a^3. Hence, we predict whether the value of a is '0' or '1'. If the prediction has been determined, subtraction process will be conducted. The next pattern onward is the factor generation that complies to $3a^2b \,(+)\, 3ab^2 \,(+)\, b^3$. This process is part of iterative calculation. The value of a and b are not the same with the previous a for MSB calculation or b for LSB calculation. Thus, the index notation of a and b are different for each stage in generated factor calculation. In the last iterative calculation of $3a^2b \,(+)\, 3ab^2 \,(+)\, b^3$ will contain b value for LSB calculation. From those processes, we will get the answer Q from cumulative guessing bits and remainder R as the last result of subtraction.

B. Optimized Algorithm

From equation (8), we can see that there are two parts that can be optimized. First one is the a^3 part and second one is the ($3a^2b \,(+)\, 3ab^2 \,(+)\, b^3$) part. For the first one, a^3 can be optimized as single a. It is because of the probability of a value are only '0' or '1'. Hence, in binary perspective $a^3 = a$. For the second one, $3a^2b \,(+)\, 3ab^2 \,(+)\, b^3$ can be optimized as $3a^2b \,(+)\, 3ab^2 \,(+)\, b^3 = 3ab\,[a \,(+)\, b] \,(+)\, b^3$ since the $(+)$ operation condition can be established in hardware architecture easily. By using those two optimizations, we will get an optimized calculation as (9) - (10).

$$a^3 = a \quad (9)$$

$$3a^2b \,(+)\, 3ab^2 \,(+)\, b^3 = 3ab\,[a \,(+)\, b] \,(+)\, b^3 \quad (10)$$

If we observe (9) and (10), we will see that there are two hardware implementations of factor generators for (9) and (10). But, we can optimize these two factor generators into a single architecture. If we observe Fig. 2, the first stage prediction is

done for determining value of a and the second stage prediction onward are done for determining b. It is because from the second prediction onward, a_n can be computed as 2 ($a_{n-1} + b_{n-1}$). Hence, we can merge the architectures for (9) and (10) become single architecture that represents $3ab\ [a \,(+)\, b] \,(+)\, b^3$, since the first prediction is place on b and initial value of $a = 0$. For example, if we predict that $a = 0$ in the first stage prediction, we will get $3ab\ [a \,(+)\, b] \,(+)\, b^3 = 3\text{x}0\text{x}0\ [0 \,(+)\, 0] \,(+)\, 0^3 = 0$. On the other hand, if we predict that $a = 1$ in the first stage prediction, we will get $3ab\ [a \,(+)\, b] \,(+)\, b^3 = 3\text{x}0\text{x}1\ [0 \,(+)\, 1] \,(+)\, 1^3 = 1$. It is exactly the same with a itself. After those three optimizations, we can achieve an optimized hardware algorithm for cube root calculation that can be written as following pseudocode.

proposed cube root algorithm

```
definition
   D     input data;
   N     input bit-width;
   Q     result;
   d     data accumulator;
   f     generated factor;
   r     remainder;
   a     cube root variable a;
   b     cube root variable b;
   iH    higher bound index;
   iL    lower bound index;
   i     iteration process index;
initialization
   a = 0;
   b = 0;
   iH = N - 1;
   iL = N - 3;
   d = D[iH : iL];
process
for (i = 1 to N/3) then
    b = 1;
    f = 3ab (a + b) + b;
    if (f <= d)
       r = d - f;
    else
       r = d;
       b = 0;
    end if
    Q = {Q[(N/3) - 1], b};
    a = 2 (a + b);
    d = {r, D[(iH - 3i : iL - 3i]};
end for
```

IV. Proposed Architecture

After optimizing the algorithm, we can design an optimized architecture for cube root calculator. There are five functional blocks to establish cube root calculator: main datapath, factor generator, control, sign-in, and sign-out units. Proposed main datapath architecture is shown in Fig. 3. In this architecture, components usage costs 3 buffer registers, 2 multiplexers, 1 concatenation, and 1 subtraction. It shows that main datapath is efficiently designed by using simple operations. For factor generator unit, proposed architecture is shown in Fig. 4. Its components usage cost 1 buffer register, 2 shifters, 6 multiplexers, 3 addition, and 1 multiplication. Multiplication is the most complex and area consuming operation here. Hence, we choose iterative process architecture in order to optimize the area consumption.

Main datapath and factor generator units are connected each other through generated factor signal. In Fig.3, generated factor signal has $n+1$ bit-width. It needs to be compared with the existing accumulated data from accumulator register. It is produced from factor generator unit in Fig. 4, which is illustrated as output signal. After the main datapath and factor generator units are succesfully designed, we need to add three more units: control, sign-in, and sign-out units.

Sign-in unit is responsible to check the input data format, sign integer has positive and negative values. In order to fulfill these requirements, we design sign-in unit to check if the input data is positive or negative. If input data is in positive form, it will be passed through to main datapath unit directly. If input data is in negative form that complies to 2's complement format, it will be converted to positive form first. Sign-out unit is responsible to convert the obtained result into valid format. If the result has to be positive, it will be passed through output directly. If result has to be negative, it will be converted to 2's complement format first before passed through output. Control unit is responsible to control the flow of cube root computation. Integrating all these units will establish a complete cube root calculator architecture as shown in Fig. 5. Actually, proposed architecture is generic architecture for integer cube root calculation. Hence, this architecture dan be adapted to comply any number of input bit-width scalability (e.g 16-bit, 32-bit, 64-bit, 128-bit, etc). In this research, we use 32-bit input for prototyping purpose.

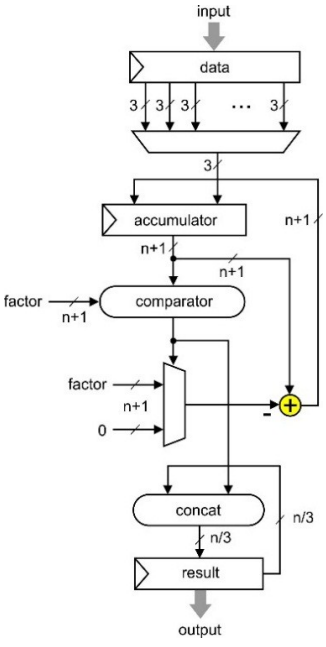

Fig. 3. Main datapath architecture

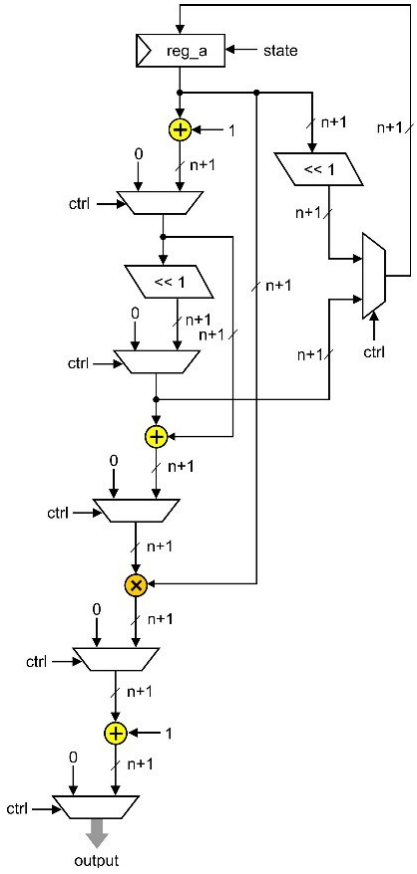

Fig. 4. Factor generator architecture

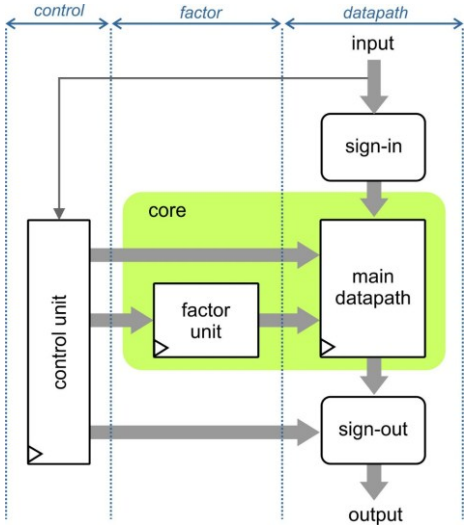

Fig. 5. Complete architecture for cube root calculator

V. Evaluation and Analysis

A. Functional Test Evaluation

For functional test purpose, we implement the proposed cube root architecture for 32-bit input bit-width by RTL coding and simulate it in Modelsim software. Fig. 6 shows the simulation result. We can see that integer input data can be computed successfully and valid output can be achieved after several clock cycles. For positive integer example, input data value is 75366 and valid output is 42, because $75366 = 42^3 + 1278$; input data value is 1730482 and valid output is 120, because $1730482 = 120^3 + 2482$. For negative integer example, input data value is -62976064 and valid output is -397, because $|-62976064| = |-397^3| + 405291$. Hence, we can conclude that proposed design is valid.

Fig. 7 shows the latency cost. We can see in the waveform, calculation process needs 13 clock cycles latency. It means that a 32-bit input data design will consume 13 clock cycles latency. Input bit-width N is 32-bits. Thus, it needs to be appended becomes 33-bits in order to accommodate 3-digit subgroup system. Thus, latency cost will be [(32+1)/3]+2. We can state it in general form as [(N+1)/3]+2.

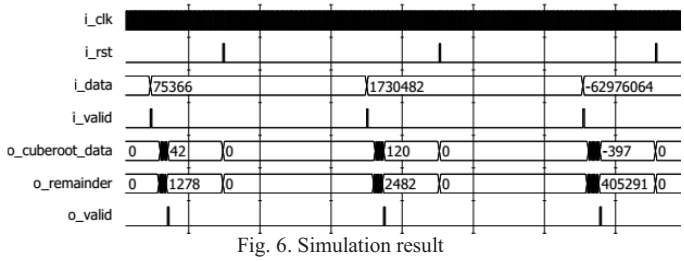

Fig. 6. Simulation result

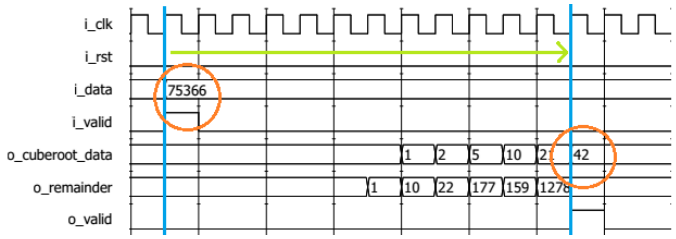

Fig. 7. Latency cost

B. Synthesis Report

The design synthesis has been conducted for $N = 32$. Table I shows the synthesis results for a 32-bit input design architecture. Synthesis processes are done for several implementation FPGA type. The results show that the design consumes small area and has promising speed. The worst case delay happens from register in factor generator to remainder register in main datapath. Thus, for pipelining strategy, we can insert a pipeline into this path to reduce worst case delay.

TABLE I. Synthesis Results

	Altera		Xilinx	
Family	***Cyclone II***	***Stratix II***	***Spartan 6***	***Virtex 5***
Area	429 LEs	288 ALUTs	415 LUTs	380 LUTs
	121 Reg	121 Reg	121 Reg	119 Reg
Delay (ns)	18.32	13.73	24.94	14.49
Freq (MHz)	54.60	72.81	40.09	69.01

C. Benchmarks

Comparison with other works might be hard to do, because the publications about the cube root FPGA implementation are hard to be found. In several cases, the existing publications can

not be compared to each other directly. Because, each paper usually has a specific issue for the calculation or its target application. For example, paper [1] proposed a cube root architecture for IEEE 754-2008 standard. Although it has the same purpose to calculate cube root value, but the target application is different to ours.

VI. Conclusion

An optimized integer cube root algorithm and its efficient architecture for FPGA implementation are presented in this paper. Integer cube root calculation is computed by using 3-digits of binary number and iterative calculation. Proposed algorithm's idea is reducing computational complexity in factor generator. For design evaluations, we design a 32-bit integer cube root architecture. Evaluation results show us that the design architecture is valid. The design latency is defined by (N/3)+2, with N is bit-width of the design input. The design also has been synthesized for several FPGA with promising results in area consumption and speed.

Acknowledgment

This research is supported by Integrated Circuit (IC) Design Laboratory, Microelectronics Center (PME), Institut Teknologi Bandung, Indonesia.

References

[1] C.M. Guardia and E. Boemo, "FPGA implementation of a binary32 floating point cube root," Proc. of Southern Conf. on Programmable Logic, pp.1-6, November 2014.
[2] J.-A. Pineiro *et al.*, "A radix-2 digit-by-digit architecture for cube root," IEEE Trans. on Computers, vol. 57, pp.562-566, April 2008.
[3] H. Peng, "Algorithms for extracting square roots and cube roots," Proc. of Symp. on Comput. Arithmetic, pp.121-126, May 1981.
[4] P. Montuschi *et al.*, "A digit-by-digit algorithm for mth root extraction," IEEE Trans. on Computers, vol. 56, pp.1696-1706, December 2007.
[5] A. Vazquez and J.D. Bruguera, "Composite iterative algorithm and architecture for q-th root calculation," Proc. of Symp. on Computer Arithmetic, pp.52-61, July 2011.
[6] S. Aslan *et al.*, "A high-level synthesis and verification tool for application specific k^{th} root processing engine," Proc. of Int. Midwest Symp. on Circuits and Systems, pp.1051-1054, August 2013.
[7] M. Ramalatha *et al.*, "A novel time and energy efficient cubing circuit using vedic mathematics for finite arithmetic," Proc. of Int. Conf. on Advances in Recent Technologies in Communication and Computing, pp.873-875, October 2009.
[8] R.V.W. Putra, "A novel fixed-point square root algorithm and its digital hardware design," Proc. of Int. Conf. on ICT for Smart Society, pp.1-4, June 2013.
[9] R.V.W. Putra and T. Adiono, "A register-free and homogenous architecture for square root algorithm," Proc. of Int. Conf. on Computer, Control, Informatics and Its Applications, pp.64-68, October 2014.