AN FPGA IMPLEMENTATION OF A FIXED-POINT SQUARE ROOT OPERATION

K. Piromsopa , C. Aporntewan and P. Chongsatitvatana

Department of Computer Engineering.
Chulalongkorn University.
254 Phayathai Road Patumwan Bangkok Thailand 10330
Phone:+66-2-218-6956, Fax:+ 66-2 -218-6955
Email:g41kpr@cp.eng.chula.ac.th , u37cap@cp.eng.chula.ac.th, prabhas@chula.ac.th

ABSTRACT

Square root operation is considered difficult to implement in hardware. In this paper, we present an FPGA implementation of a 32-bit fixed-point square root based on the non-restoring square root algorithm. The operation latency is 25 clock cycles (with 8-bit floating-point precision). The circuit occupies 161 Logic Cells (13%) of Altera Flex 10K20RC240-4 FPGA. The maximum clock cycle archived is 21.36 MHz. It means that the solution of a 32-bit fixed-point number can be calculated in about 1170 nanoseconds.

1. INTRODUCTION

Square root is a basic operation in computer graphics and scientific calculation applications. Because of the complexity of the square root algorithms, the square root operation is hard to implement on hardware. In this paper, we explain various square root algorithms and our choice of the one that suits for implements on an FPGA.

2. SQUARE ROOT ALGORITHMS

There are many algorithms and implementations of square root on VLSI and FPGA. Three algorithms will be discussed: Newton-Raphson method [1] [2] [3], SRT-Redundant method [4] [5], and Non-Redundant method [6] [7]. These three algorithms will create an approximate value of the solution. However, Y. Li and W. Chu have introduced a new Non-Restoring Square Root Algorithm [8] [9], which will generate an exact resulting value.

2.1 The Newton-Raphson Algorithm.

The Newton-Raphson method was first used in Cray-2 and Burroughs BSP. Iterative methods start with an initial (guess) value and improve accuracy of the result with each iteration. Assuming that X is the original number, the iterative equation for calculating the reciprocal of its square root, Y, is (1).

$$Y_{i+1} = Y_i (3 - X (Y_i)^2) \qquad (1)$$

Once Y have been calculated, one can get the square root by multiplying with X. This algorithm has quadratic convergence.

The algorithm needs a seed generator for generating Y_0(the initial guess value of the result), using a ROM table for instance. The number of entries in the ROM and the length of each entry affect the precision of the operation and the number of iterations. To get the 24-bit result, the ROM should be 64 words x 6bits. With this (pseudo) 8-bit guess value only two iterations are necessary for 24-bit precision. In each iteration, multiplications and additions or subtractions are needed. In order to speed up the multiplier, there must be a special algorithm such as Wallace tree to get a partial production and use a carry propagate adder to get the production. Because the multiplier requires a rather large number of gate counts, it is not so practical to place many multipliers on FPGA. Also, it is hard to get an exact remainder of square root.

2.2 The Radix-2 SRT-Redundant and Non-Redundant Algorithm.

The Radix-2 SRT-Redundant method and Non-Redundant method are similar. Since they both based on recursive relation. In each iteration, there will be one digit shift left and addition. The determination of a function is rather complex, especially for high-radix SRT algorithm. The implementations are not capable of accepting a square root on every clock cycle. Also notice that this two methods may generate a wrong resulting value at the last digit position.

2.3 The Non-Restoring Algorithm.

The non-restoring Square Root Algorithm uses the two's complement representation for the square root result. At each iteration the algorithm can generate exact result value even in the last bit. There is no need to do the complex calculation as appear in the SRT-Redundant and other methods. The exact remainder can be obtained immediately (with a little correction if it is negative)

Assume that the radicand is an 32-bit unsigned number (denoted by D31..0). The square root for a 32-bit radicand is denoted by Q15..0. R is the remainder $(R=D-(Q)^2)$ which will be denoted by R16..0. Since this is redundant representation for square root, exact bit can be obtain in each

iteration. The non-restoring square root algorithm is given below (Fig. 1)

```
Let
    D be 32-bit unsigned integer
    Q be16-bit unsigned integer (Result)
    R be 17-bit integer (R = D – Q²)
Algorithm
Q = 0;
R = 0;
for  i = 15 to 0 do
    if (R >= 0)
        R = (R << 2) or (D >> (i + i) & 3);
        R= R - ((Q << 2) or 1);
    Else
        R = (R << 2) or (D >> (i + i) & 3);
        R= R – ((Q << 2) or 3);
    End if
    if (R >= 0) then
        Q = (Q << 1) or 1;
    Else
        Q = (Q << 1) or 0;
    End if
```

Fig. 1 Non-Restoring Square Root Algorithm

Let us see an example in which D is an 8-bit radicand value 140 (10001100_2). The 4-bit solution Q should be 11 (1011_2) and remainder should be 19 (10011_2).

```
Set Q = 0000, R = 000000
i=3:
R>=0, R = 000010 – 000001 = 000001
R>=0, Q = 0001
i=2:
R>=0, R = 000100 – 000101 = 011111
R<0,  Q = 0010
i=1:
R<0,  R = 011111 + 001011 = 001010
R>=0, Q = 0101
i=0;
R>=0, R = 101000 – 010101 = 010011
R>=0, Q = 1011
```

To correctly determine the value of R, we need to add 1 more extra bit to R (Consider as sign bit).

In order to get more precision digit, you can simple shift left the radicand by 2n before the iteration. And shift right the solution by n to get the correct answer. For example if you want to find the square root of 35 (00100011_2) with one binary floating point. You have to find the square root of 10001100_2.(Shift left by 2 bits) After you get the answer (1011_2) you will have to shift right by 1 bit.

So the correct answer is 101.1_2 or 5.5.

3. HARDWARE DESIGN

Since the algorithm of Non-Restoring Square Root show only a few functions (only Addition, Subtraction, Shift Left, and Shift Right). We decided to use this one. (Newton-Raphson method require a ROM and multiplier which is not so practical to implement on a small FPGA)

We design the circuit by using two shift registers (one is shift 1-bit left, the other is shift 2-bit left), one normal register, and an adder. D is the Radicand, Q is the Solution, and R is the Partial Remainder.

This design is different from the design of [9], our aim is to avoid structural description and rely on the synthesis tool to accept our behavioral description and generate a good circuit.
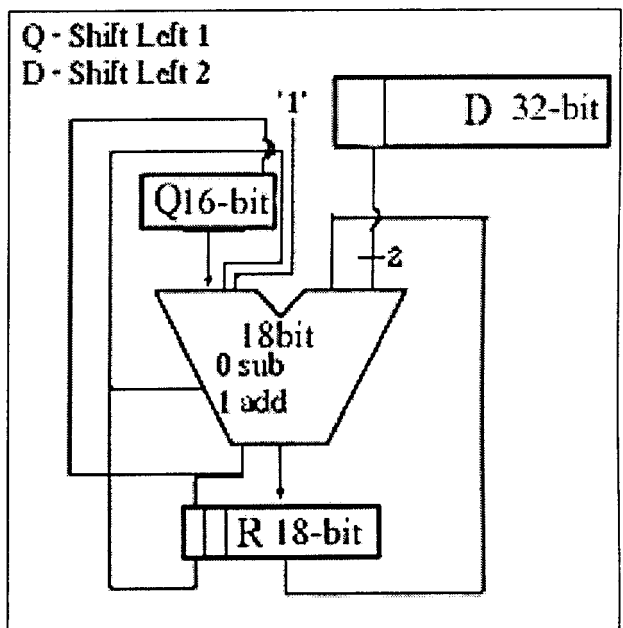

Fig. 2 Design of Non-Restoring Square Root

The size of each register (D, Q and R) and ALU can be determined by the size of Radicand register. If the Radicand contain X bit. Q will be X/2 while ALU and R would be (X/2)+2 bit. And the total number of iteration is (X/2)+1 cycle. In each cycle D will shift left 2 bit and Q will shift left 1 bit. The start up value of Register Q and R is 0 and should be clear once the radicand is loaded into register D.

For example if you want to find square root of a 32-bit integer with 8-bit floating-point, you will have to find square root of 48-bit radicand. Then the size of D will be 48 bit. The solution will be 24 bit (16-bit integer with 8-bit floating-point). The registers R and ALU will be operate at 26 bits. While the circuit will need 25-clock cycles for the solution.

4. PERFORMANCE ANALYSIS

Design Entry of this circuit is written in VERILOG HDL. Altera FLEX10K20RC240-4 FPGA can operate the 48-bit radicand square root circuit at 21.36 MHz.. Comparing to the same software algorithm written in about 2000 lines of assembly code on a custom processor with no floating-point unit (average CPI is 4), the square root circuit consumed 13% of Logic Cells. (see Table. 1)

Implementation	Time (Nanoseconds)
Software with 2000 lines of assembly code operate on custom CPU at 25 MHz	320000
Hardware operate at 20 MHz	1250

Table 1. Execution time analysis.

The speed up is 32000/1250 = 256 times faster than using pure software.

After the square root circuit has been tested, it is embedded in to a core processor. The Altera Max + plus II reports the critical path in the control path. This is unexpected since we normally suspect the ALU to be the bottleneck. We have not done any further optimization on the control unit.

5. CONCLUSIONS

There are a lot of Square Root Algorithms today. On the systems that uses square root operations frequently, to archive high performance at a reasonable cost, the appropriate algorithm must be implemented.

The Non-Restoring Square Root Algorithm can be implemented with fewest and the result is the fastest circuit among Newton-Raphson and Non-Redundant methods. To meet the higher performance the fully pipelined functional unit can be used. From our experience, on an FPGA, implementing your own Carry Look Ahead adder circuit might not give you better performance, since the synthesizer often achieves an impressive performance.

REFERENCES

[1] J. Hennessy and D. Patterson: "Computer Architecture A Quantitative Approach," Second Edition, Morgan Kaufmann Publishers, Inc., 1996. Appendix A: Computer Arithmetic by D. Goldberg.

[2] C. Ramamoorthy, J. Goodman, and K. Kim: "Some properties of iterative Square-Rooting methods Using High-Speed Muliplication," IEEE Transaction on Coputers, Vol. C-21, No. 8, 1972. pp837-847

[3] H. Kabuo, T.Taniguchi, A. Miyoshi, H. Yamashita, M.Urano, H. Edamatsu, S. Kuninobu: "Accurate Rounding Scheme for the Newton-Raphson Method Using Redundant Binary Representation," IEEE Transaction on Computers, Vol.43, No. 1, 1994. pp43-51

[4] M. Birman, A. Samuels, G. Chu, T. Chuk, L. Hu, J. McLeod, and J. Barnes, "Developing the WTL3170/3171 Sparc Floating-Point Coprocessors," IEEE MICRO February, 1990. pp55-64.

[5] M. Ercegovac and T. Lang: "Radix-4 Square Root Without Initial PLA", IEEE Transaction on Computers, Vol.39. No.8, 1990. pp1016-1024.

[6] J.Bannur and A. Varma, "The VLSI Implementation of A Square Root Algorithm," Proc. of IEEE Symposium on Computer Arithmetic, IEEE Computer Society Press, 1985. pp159-165.

[7] K. C. Johnson: "Efficient Square Root Implementation on the 68000," ACM Transaction on Mathematical Software, Vol. 13, No. 2, 1987. pp138-151.

[8] J. Bannnur and A. Varma: "The VLSI Implementation of A Square Root Algorithm," Proc. of IEEE Symposium on Computer Arithmetic, IEEE Computer Society Press, 1985. pp159-165.

[9] Y. Li and W. Chu: "Implementation of Single Precision Floating Point Square Root on FPGAs," Proc. of the 5th IEEE Symposium on FPGA-Based Custom Computing Machines (FCCM '97), IEEE Computer Society, 1997. pp226-232