DIGITAL SYSTEMS FOR SQUARE ROOT COMPUTATION
Borisav Jovanović, Milunka Damnjanović, Faculty of Electronic Engineering Niš

1. INTRODUCTION

Digital systems for square root computation and division are still a challenge for IC designers. There are many algorithms and their implementations. A unit with 24-bit input is considered here. Two optimization criteria were used: chip area and speed. Therefore, the number of iterations for square root computation should be minimal.

Attention has been paid to three types of square-rooters. Each system is considered starting with mathematical proof of the algorithm. After that, exact hardware implementation has been developed. All systems were verified through the VHDL simulations and synthesized by Cadence tools.

In the next, three considered algorithms and their implementations will be described and analysed.

2. NEWTON-RAPHSON’S METHOD

Following Newton-Raphson’s method (also known as Heron’s)[2], the square root value of number \( a \) is computed through the iterative formula

\[
x_{k+1} = \frac{1}{2} \left( x_k + \frac{a}{x_k} \right)
\]

Digital system of this formula implementation is shown on Fig. 1.

![Fig.1. Block diagram of digital system for square-rooting based on Newton-Raphson’s method](image)

It is built of few subsystems: 24-bit register U2 used for storing the number that square-root should be computed for; 13-bit register U1 in which temporary solution for square root is stored; subsystem U8 for the division implementation; subsystem U5 for initial solution-value computing; adder U3 containing 13 full adders; counter_2 that counts started divisions; sequencer U6 that has the control over subsystems. The system ports are: signal \( clk \) for clock; \( start_{root} \) for beginning of computation; \( end_{root} \) for the indication of the operation completion, bus \( a(23:0) \) for getting the input number \( a \); bus \( root(12:0) \) for output data.

How does the entire system function? At the beginning, number \( a \), the dividend, is stored in register U2. Unit for initial value computation, U5, gets the value \( a \) from register U2 and after time delay it produces initial solution \( x_0 \). After completion of generating \( x_0 \), this value is available on the output of register U1 as divisor. Subsystem for dividing divides values stored in registers U2 and U1, and quotient is added to the value stored in register U1 and shifted right. After these operations, new solution \( x_1 \) is stored in register U1. Only one additional dividing operation is needed to reach the final value of square root, because the initial value \( x_0 \) is computed on appropriate way, instead assumed as the arbitrary value. In second dividing operation, \( x_1 \) is divisor. Quotient is added to \( x_1 \), shifted right and stored in register U1. This final result can be read on bus \( root(12:0) \).

Further, the attention will be paid on subsystems implementing the parts of square-rooter.

The computation of the initial value \( x_0 \) of square root is done with the subsystem shown in Fig.2.

![Fig.2. Block diagram of digital subsystem for initial square-root-solution generation](image)

Its inputs are: \( clk \) for clock, \( start_{gen} \) for operation start, \( end_{gen} \) for signaling if unit finished its operation, 24-bit input bus \( a(23:0) \) and output bus \( x0(12:0) \). It consists of one shift register U1, counter U0 and few simple logical gates. Input number \( a \) (for which the square root should be computed) is stored in 24 least significant bits of 25-bit register U1. At the start of computation, the number of significant digits of the input number is unknown and therefore it should be computed first: counter U0 counts the number of zeros before the most significant 1 in the input number \( a \).

After that, number \( a \) is stored again in register U1. The number of digits in initial solution is twice less than the number of input digits and it is stored in counter. Register U1 starts shifting to the right and counter starts decrementing its
value. When the counter reaches zero state, shifting is stopped and the value remained in $U1$ is initial solution for square root.

The number of necessary clock iterations for initial value generation $N_{gen}$ is:

$$N_{gen} = 25 - \left\lfloor \frac{K - 1}{2} \right\rfloor$$

where $K$ is the number of significant bits of input $a$. $N_{gen}$ is in the range from 25, for computing square root of 1, to 14, for computing square root of $(FFFFFFF)_{16}$.

The division unit implements the manual or Longhand division algorithm [4]. Its structure is shown in Fig.3.

![Fig.3. Block diagram of division implementation subsystem](image)

Fig.3. Block diagram of division implementation subsystem

Dividend is 24-bit number, divisor is 13-bit number and quotient is 13-bit number. At the beginning of dividing procedure, RegisterB (initially storing divisor) is shifted to the left until its value becomes greater then value stored in RegisterA (dividend). The 5-bit counter counts the number of shifts. Then, counter starts counting down, and the procedure of quotient-digits computing follow. RegisterA is shifted left for one bit. Further, RegisterA changes its value, but RegisterB is keeping its value. The dividing procedure is performed through the successive subtractions (RegisterA - RegisterB). Each time the result is negative, the value RegisterA is shifted left for one position and zero is appended to RegisterC value on least significant position. Else, result of subtraction, shifted left for one position, is stored in RegisterA and digit 1 is appended to RegisterC. Since the number of quotient digits is equal to the number of shifts, the division operation is finished when counter reaches zero. The number of clock cycles needed for dividing is two times greater then number of significant digits of the quotient.

The total number of clock periods (for initial value generation) is:

$$N_{gen} = 25 - \left\lfloor \frac{K - 1}{2} \right\rfloor$$

Obtained results confirm the idea that traditional methods are usually the best ones. The algorithm is very fast and its hardware implementation has small chip area. The algorithm computes square root on the same way like people do manually:

It starts with grouping the digits in pairs, starting from the decimal point. The first digit of result is the greatest digit (e.g. $b_n$) whose square is less than the first group value. The positive remainder after square of $b_n$ has been subtracted from the first group value, should be concatenated with next pair of digits and treated integral (e.g. value $c_m$). Present result is formed of found digits (at the beginning it is only $b_n$). Next digit (e.g. $b_{m-1}$) in the resulting number, is the greatest number that meets the condition

$$(20 \times \text{present_result} + b_{m-1}) \times b_{m-1} \leq c_m$$

The result of subtraction

$$c_m = 20 \times \text{present_result} + b_{m-1} \times b_{m-1}$$

should be concatenated with next pair of digits (like previously) and treated integral (e.g. value $c_{m-1}$) hereafter. Present result is appended by new digit $b_{m-1}$. The procedure is repeated until all input groups of digits are considered.

![Fig.4. Block diagram of square rooter based on iterative algorithm](image)

Fig.4. Block diagram of square rooter based on iterative algorithm

Computation is significantly simpler if it is performed in radix 2. Like previous computation in radix 10, binary digits of number which square root should be found are grouped in pairs. In every iteration, two digits are appended to the right side of the present result (digit 0 for multiplying by two and digit 1 for possible digit of result). In radix 2 only digits 1 and 0 can be assumed for the next digit of result. Subtraction is performed, and if minuend is greater then subtrahend, digit 1 is appended to present result. Result of subtraction, concatenated with next pair of digits, form minuend for next subtraction. Else, present result is concatenated with zero. Minuend concatenated with next pair of digits forms minuend for next iteration.

Schematic of digital system implementing iterative algorithm is given in Fig.4. System ports are: clk for clock, start_root for computation beginning, input 24-bit bus $a(23:0)$ for input data, output 11-bit bus $root(11:0)$ for output data, output signal end_root for indication that computation is finished. It consists of 36-bit register Reg1, 12-bit register Reg2, subtractor Sub1 that consists of 14 full adders and 14 inverters, 4-bit counter Counter_4 and few simple logic gates.

At the beginning, 24 least significant bits of Reg1 get the input value $a$. Other 12 most significant bits are set to zeros. Reg2 also gets zero value. Minuend is made of 14 most
significant bits of Reg1. Present result is stored in Reg2. At the beginning of computation, present result is zero. Subtrahend is made of present result and two binary digits, zero or one, appended to the right side of present result. If minuend is greater than subtrahend, present result is appended by binary digit one. Difference is stored on minuend’s place, in 14 higher bits of Reg1. After that, Reg1 is shifted left for two positions. If minuend is smaller than subtrahend, subtraction results are not stored. Reg1 is just shifted left for two positions.

The algorithm is very fast. It uses only 12 iterations for the square-root completion. Every iteration is completed exactly in one clock period.

4. BINARY SEARCH ALGORITHM

This algorithm can be found solving many different tasks in programming. Here, this algorithm is modified to speed up root computation [6]. The computing procedure is simple:

The resulting square root value, e.g. \( a_m a_{m-1} a_{m-2} ... a_1 a_0 \), has twice less number of bits comparing to input number, 12 bits in our case. The algorithm finds the value of square root bit-by-bit. First, the most significant bit \( a_m \) is assumed to be 1 and other bits are zeros. Number 100...0 is squared and subtracted from the input number. If the remainder is same number of inverters, gives the control signal \( \text{sel} \) for carry out output. If minuend is greater then subtrahend then \( \text{sel} \) is 1 and difference should be loaded into Reg1. Else, \( \text{sel} \) is 0 and Reg1 retains its old value. Negative data should not be stored.

Adders Add1 and Add2 are composed of logical OR gates. They get values of Reg2 and Reg3 outputs on their inputs. One of them provides subtrahend(23:0) on its output bus:

\[
\text{subtrahend}(23:0) = \text{Reg2} + (\text{Reg3})^2 = 2^k A + 2^{2k-2} (6)
\]

The other provides the data input for Reg2:

\[
\text{Reg2} = \text{Reg2} >> 1 + (\text{Reg3})^2 = 2^{k-1} A + 2^{2k-2} = 2^{k-1}(A + 2^{k-1}) = 2^{k-1}(a_m a_{m-1} a_{m-2} ... a_k 10...0) = 2^{k-1} A' (7)
\]

The >>1 denotes that Reg2 value is shifted one bit to the right. There is no carry transition, so OR logical gates are used instead of full adders, providing significant saving in chip area. Number Reg2>>1 = 2^k A >> 1 has 2k-1 zeros at least significant positions, while \( (\text{Reg3})^2 = 2^{2k-2} \).

In every iteration, subtraction is performed. Subtrahend, \( 2^k A + 2^{2k-2} \), is subtracted from minuend, Reg1 value (B-A'). If minuend is greater than subtrahend, guess \( A' \) is correct and digit 1 is correctly assumed on bit position k-1. New temporary solution get value from previous guess A'. New values are stored in Reg1 and Reg2:

\[
\text{Reg1} := (B - A') - 2^k A + 2^{2k-2} = B - (A + 2^{k-1}) = B - A' (8)
\]

\[
\text{Reg2} := 2^{k-1} A' (9)
\]

If minuend is less than subtrahend, guess \( A' \) is incorrect and digit 1 is not correctly assumed on bit position k-1, i.e., 0 is the solution for bit position k-1. Temporary solution A retains its value, so Reg1 value is the same as previous one. Value in Reg2 is shifted right for one bit position:

\[
\text{Reg2} := (\text{Reg2} >> 1) = (2^k A >> 1) = 2^{k-1} A (10)
\]
After 12 \((m+1=12)\) clock iterations, \(\text{Reg2}\) holds correct value of square root which can be read on bus \(\text{root}(11:0)\)

\[
2^k A = 2^k A = a_m \ a_{m-1} \ ... \ a_1 \ a_0
\]

The implementation is verified in VHDL, and synthesized in Cadence PKS logical synthesis tool. After that, standard cell net list was put into program Silicon Ensemble where placement and routing were performed.

5. RESULTS

Comparing the implementations independently of the physical realisation, we can see that the implementation based on Newton-Raphson’s method requires greater number of clock periods (in range from 40 to 70 depending on input data) than the other methods (exactly 12 clock cycles) for completing the square root solution.

All systems are verified in VHDL and implemented in same standard cell technology. So, the properties of the resulting solutions can be regularly compared.

Namely, the first VHDL simulation was performed in ActiveHDL tool. Logical synthesis is done in BuildGates (part of Cadence). Alcatel 0.7\(\mu\)m CMOS standard cells technology has been chosen. As a result, logical synthesis produced standard cell net list that was brought to another Cadence program, Silicon Ensemble, where placement and routing were performed. Extracted signal propagation delay values and standard cell net list were put back into simulator NCsim where logical simulation was performed. Waveforms derived from NCsim matched with ones from ActiveHDL. Finally, layouts were verified by Design Rule Check analysis.

As an example, layout derived from Silicon Ensemble for the iterative algorithm based square root implementation is shown in Fig. 5.

![Fig.5. Layout of digital system for square rooting made following iterative method](image)

Some properties of the considered implementations are compared in Table 1. Maximal clock frequency is determined by maximal propagation delay in adders and subtractors. All proposed systems have incorporated adders with serial carry transition. If other types of adders were used, both occupied chip area and maximal clock frequency would be greater.

6. CONCLUSION

Three square root implementations were designed and their properties were compared. System implementation based on iterative algorithm provides the solution with smallest chip area and power consumption, and maximal clock frequency.

All proposed solutions are very flexible and can be modified if square root of a number with more or less than 24 bits is required, if it is required to find bits after decimal point, etc. These demands can be accomplished by minor changes in VHDL code.

It is worth to notice that Newton-Raphson's algorithm based square root computation circuit has the advantage of having built-in a division subsystem. So, it can be applied in all circumstances where both are needed, integer division and square root computation. In that case, significant saving in chip area can be accomplished.

This work was supported in part by the Ministry of Science, Technology and Development of Serbia, through the Project IT.1.02.0076.A realized in Technology Development area.

REFERENCES


Sadržaj – U radu su razmotrene implementacije tri algoritma za izračunavanje kvadratnog korena nekog broja. Implementacije Newton-Raphson-ovog, iterativnog i algoritma korenovanja binarnim pretraživanjem su projektovane, verifikovane i uporedene. Kompletno su opisani algoritmi, fizičke realizacije sistema i njihovo funkcionisanje.

**DIGITAL SYSTEMS FOR SQUARE ROOT COMPUTATION**

Borisav Jovanović, Milunka Damnjanović