

The Implementation of Square Root Circuits in ASIC

Petar Ristić, Borisav Jovanović

Abstract – This paper presents design, implementation, and comparative analysis of three circuits computing the square root of fixed-point numbers. The circuits are first described in VHDL and then, they are synthesized and implemented using Cadence tools - Genus and Innovus, in technology node TSMC 65nm. The design process starts with RTL description and ends with ASIC layout design. Performance of each circuit is evaluated across three critical metrics: computational latency, dynamic, static power consumption and occupied area. Our results reveal trade-offs between these metrics, required by different applications. The design process provides practical insights for selecting square root implementations in area-constrained and power-sensitive applications.

Keywords - square root algorithms, ASIC, TSMC 65nm, Cadence.

I. INTRODUCTION

Digital square-root computation remains a significant challenge in integrated circuit design due to strict constraints on precision, latency, and silicon area. In telecommunication applications, where digital baseband signals are processed, often it is required to find the envelope of I (in-phase) and Q (quadrature) signal components. In such applications, the square root circuit runs at high speed, producing the square root values for every sample of input I/Q signals. State-of the art RF transceivers exhibit high sample rate values, often equal to several hundreds of mega samples per second (MSps). Other examples include high speed cameras where pixel processing is also computationally intensive. In embedded systems, in those applications where square root operations are not frequently performed, area efficiency and low-complexity circuit implementation are prioritized [1].

This work investigates different hardware architectures computing the square root of fixed-point operands, with an emphasis on area minimization, while maintaining high computational performance. The architectures implement iterative radix 2 square-root algorithm, which is chosen because of deterministic iteration count and low-complexity hardware implementation.

Three square root circuit architectures are developed and evaluated: a sequential implementation, a pipelined version, which improves throughput by overlapping successive iterations, and software implementation, realized in assembler program of custom-made, simple 8-bit microcontroller. The circuits are described in VHDL and synthesized using Cadence tools Genus and Innovus.

Petar Ristić, Borisav Jovanović, Faculty of Electronic Engineering, Aleksandra Medvedeva 4, 18000 Niš, Srbija, E-mail: petar.ristic@elfak.rs, borisav.jovanovic@elfak.ni.ac.rs

Target ASIC technology node is TSMC 65nm. The functional verification is carried out using Cadence simulation tools NCLaunch and NCSim. The proposed circuits are compared in latency, area, speed, and power consumption. The results of comparison suggest most suitable square-root circuit architecture in different applications and for specific requirements.

II. SQUARE ROOT ALGORITHM DESCRIPTION

The iterative method follows the same principle found in traditional “longhand” square-root operation. When square root operation is performed in decimal radix, the decimal digits of the input operand are grouped into pairs, starting from the decimal point to the left. The square root operation is performed in iterations. One digit is found in an iteration [2].

The first digit of the result q_0 is chosen as the greatest digit $q_0 \in \{0, 1, 2, \dots, 9\}$ whose square q_0^2 does not exceed the value of the first digit group. The positive difference, obtained after q_0^2 is subtracted from the first group value, is then concatenated with the next pair of digits and treated as a remainder D_1 in the next iteration.

In order to describe iterative algorithm mathematically, for the iteration denoted with i , we define the partial result Q_i , consisting of all previously determined decimal digits $q_i, q_{i-1}, \dots, q_1, q_0$. The next digit q_{i+1} is chosen as the maximum which meets the following condition:

$$(20 * Q_i + q_{i+1}) * q_{i+1} \leq D_i, \quad (1)$$

where D_i represents the remainder from previous iteration. The result of the subtraction, given by following equation:

$$D_i - (20 * Q_i + q_{i+1}) * q_{i+1}, \quad (2)$$

is concatenated with the next digit pair, forming the next remainder D_{i+1} . The procedure is repeated until all input groups of digits are considered.

In the radix 2, the procedure for square root calculation is significantly simpler. Only digits 0 and 1 are possible $q_i \in \{0, 1\}$ and the condition (1) can be transformed to:

$$(2 * Q_i + q_{i+1}) * q_{i+1} \leq D_i, \quad (3)$$

In order to find next binary digit q_{i+1} , it is sufficient to test $q_{i+1} = 1$. Then, the following difference is calculated:

$$D_i - (2 * Q_i + 1) \quad (4)$$

If the difference, given by Eq.(4), is greater or equal

than zero, the candidate bit $q_{i+1}=1$ is adopted and the difference is concatenated with the next input digits pair to form the D_{i+1} . Otherwise, a zero is appended to Q_i and remainder D_i is concatenated with the next input pair to form the next D_{i+1} . The procedure is repeated until all binary digit pairs are processed. The number of iterations is two times less than the number of bits in the input number. For example, for 16-bit input operand, 8 iterations are required to compute the square root value [2].

III. THE CIRCUIT ARCHITECTURES

We have implemented three architectures: the non-pipelined square-root circuit (NPSR), pipelined square-root circuit (PSR) and software-based square root (SSR). All three implementations are based on iterative algorithm.

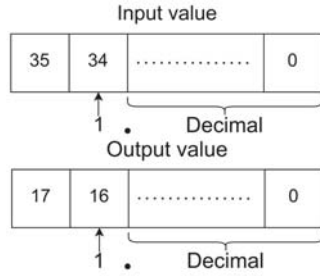


Fig. 1. Data formats for the input and output values.

The circuits calculate the square root of normalized, fixed-point 36-bit input number, where decimal point is positioned between the 34th and 33th bit. The data formats of input and output numbers are depicted in Fig. 1. The output result is 18 bits wide, with decimal point residing between bit positions 15 and 16 (Fig. 1). Although data formats provide greater ranges, the input and output values are constrained to fixed-point numbers which are less or equal than real number of 1.0.

Non pipelined square root circuit

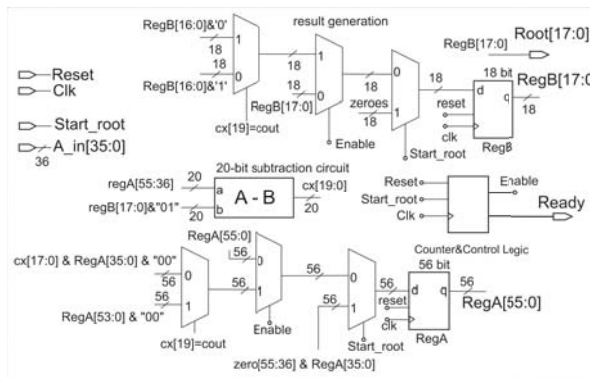


Fig. 2. Digital schematic of the non pipelined circuit

The non-pipelined square-root circuit (NPSR) produces result after 18 clock periods from the operation start. New

data cannot be applied to the input until the previous result is calculated.

The schematic of non-pipelined square root circuit is depicted in Fig. 2. The inputs are *Clk* (clock signal), *Reset* (asynchronous active-low reset), *Start_root* for calculation start and *A_in* (36-bit input operand). The output is *Root* (18-bit result) and *Ready*, indicating operation end. The schematic consists of the following main blocks:

- a 56-bit shift register RegA, which stores the input operand and also, intermediate values of remainder D_i ,
- an 18-bit register RegB, storing the partial square-root result Q_i ,
- a 20-bit subtraction arithmetical circuit, and
- control logic, including the 5-bit counter, which tracks the iteration progress.

The calculation process is triggered by signal *Start_root*='1'. At calculation beginning, the 36-bit operand is loaded via input port *A_in* (Fig. 2) into the LSB part of a 56-bit register *RegA*[35:0]. At the same time, the upper part of the register *RegA*[55:36] is filled with zeroes. Besides, at the operation beginning, the register *RegB*[17:0] is reset. The iteration counter *Cnt* is also reset and the *Enable* signal is asserted.

In each iteration, the difference is calculated, where 20 uppermost bits of RegA (*RegA*[55:36]) form a minuend D_i , and the subtrahend is comprised of the content of 18-bit RegB (partial square-root result Q_i), appended with binary sequence "01".

$$D_i - (2 * Q_i + 1) = \text{RegA}[55:36] - (\text{RegB}[17:0] * 2 + 1) \quad (5)$$

The difference is given in Fig. 2 via bus *Cx*[19:0]. If difference is positive or zero (the condition *Cx*[19] = '0' is met), the part of the result *Cx*[17:0] is loaded into 18 MSB bits of RegA, and '1' is appended to RegB:

$$\begin{aligned} \text{RegA}[55:0] &= \text{Cx}[17:0] \& \text{RegA}[35:0] \& "00" \\ \text{RegB}[17:0] &= \text{RegB}[16:0] \& "1" \end{aligned} \quad (6)$$

If difference is negative (*Cx*[19] = '1'), the RegA is just shifted left by two positions and '0' is appended to RegB.

$$\begin{aligned} \text{RegA}[55:0] &= \text{RegA}[53:0] \& "00" \\ \text{RegB}[17:0] &= \text{RegB}[16:0] \& "0" \end{aligned} \quad (7)$$

Signal *Enable* (shown in Fig. 2) serves as enable signal for RegA and RegB load/shift operations. It is active in interval of 18 clock periods. After 18 clock periods, the *Ready* flag is asserted and the final result is output on *Root*[17:0]. Table 1 provides a simplified example of circuit operation, when circuit calculates the square root of an 8-bit number (25)₁₀. The result is a 4-bit number (5)₁₀.

TABLE I
OPERATIONS OF ITERATIVE SQUARE ROOT ALGORITHM

n	operation	RegA	RegB	A-B	C
0	Load	000000011001	0000	111111	0
		000000011001	0000	111111	0
1	Shift RegA<<2	000001100100	0000	00 0000	1
		000001100100	0000	00 0000	1
2	Append A- B to RegA	000010010000	0001	11 1111	0
		000010010000	0001	11 1111	0
3	Shift RegA<<2	001001000000	0010	00 0000	1
		001001000000	0010	00 0000	1
4	Append A- B to RegA	000000000000	0101		
		000000000000	0101		

Pipelined iterative square root circuit

The pipelined square-root circuit transforms the sequential operations into parallel by replicating the computational hardware. Instead of reusing a single arithmetic operator, the pipelined design instantiates 18 identical processing stages, connected in cascade. Each stage contains unique arithmetical operator.

The pipelined circuit produces a new result every clock cycle. This means that for a new input, the result is obtained with a delay of 18 clock cycles, and, input data can change in every clock period. Although PSR and NPSR exhibit identical latency of 18 clock cycles for a single input, the pipelined version achieves 18 times higher processing throughput.

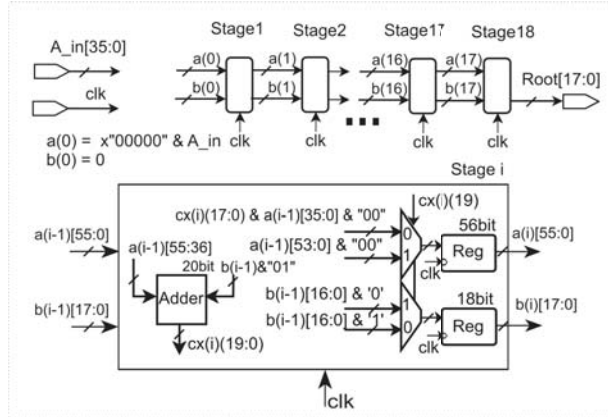


Fig. 3. Block diagram of the pipelined iterative circuit

The pipelined square root schematic is given in Fig 3. The inputs are *Clk* (clock signal) and *A_in* (36-bit input operand). The output is *Root* (18-bit result).

The circuit consists of 18 cascaded processing stages, where each stage contains:

- 56-bit shift register *A(i)*,
- 18-bit partial result register *B(i)*,
- a 20-bit arithmetical circuit used for subtraction.

At stage 0, 36-bit input operand is stored in 36 LSB bits of register *A(0)* via input port *A_in*. Upper bits of *A(0)* are reset. At the same time, the register *B(0)* content is reset.

At each stage *i*, new difference is calculated, where 20 MSB bits of *A(i-1)* – the bit positions *A(i-1)[55:36]* form the minuend *D_{i-1}*. The subtrahend consists of the content of 18-bit register *B(i-1)* (partial square-root result *Q_{i-1}*), appended by "01" binary sequence

$$Cx(i) = D_{i-1} - (2 * Q_{i-1} + 1) = A(i-1)[55:36] - (B(i-1)[17:0] * 2 + 1), \quad (8)$$

The difference is given in Fig. 3 via bus *Cx(i)[19:0]*. If difference is positive or zero (when *Cx(i)[19] = '0'*), the part of the result *Cx[17:0]* is loaded into 18 MSB bits of *A(i)*. The other bits of *A(i)* are formed of *A(i-1)*, shifted left by two bit positions. The *B(i)* is formed of *B(i-1)*, shifted left by one bit position, while its LSB bit is set.

$$A(i)[55:0] = Cx(i)[17:0] \& A(i-1)[35:0] \& "00" \\ B(i)[17:0] = B(i-1)[16:0] \& "1" \quad (9)$$

If difference is negative (*Cx[19] = '1'*), the *A(i)* is determined as *A(i-1)*, shifted left by two positions. The *B(i)* is obtained as *B(i-1)* shifted left by one bit position.

$$A(i)[55:0] = A(i-1)[53:0] \& "00" \\ B(i)[17:0] = B(i-1)[16:0] \& "0" \quad (10)$$

Software-based square root

Unlike previous two methods that are implemented in hardware, this method is implemented in software and thus requires us additional components to execute. This implementation was done on a custom built 8-bit microcontroller. In the example, we have calculated square root of 16-bit number and got an 8-bit output.

The microcontroller architecture comprises a microprocessor core, interfaced to program and data memory, and peripheral modules via shared internal data and address buses. The core includes an 8-bit Arithmetic Logic Unit (ALU), general-purpose registers, and control logic for instruction sequencing. The core executes single assembler instruction in four clock cycles.

The software implementation follows the same iterative radix-2 algorithm described in Section II, but executes it sequentially on the microprocessor core. The input operand is stored in data memory, occupying multiple consecutive memory locations. Although the implementation processes a 16-bit input to produce an 8-bit output, the methodology is similar for any number of bits.

The execution begins with instructions fetch from program memory. The registers, which are implemented of flip flops in the hardware versions, are now realized as variables located in data memory. In each iteration, the core performs the following operations: it loads the current

remainder and partial result from memory locations, executes the subtraction using ALU, evaluates the overflow bit to determine the next result bit, and stores updated values back to memory.

```
def sqrt(input_16bit):
    # Initialize
    A = input_16bit
    B = 0
    # 8 iterations
    for i in range(8):
        # Construct the operands
        Ax = (A >> 14) & 0x3FF
        Bx = (B << 2) + 1
        # Test subtraction
        C = Ax - Bx
        if C < 0: # Overflow (MSB set)
            # Don't update A_H, just shift
            A = (A << 2)
            B = (B << 1) # Insert 0
        else:
            # Update with new remainder
            A = ((C << 14) | (A << 2))
            B = ((B << 1) | 1) # Insert 1
    return B
```

Fig 4. Pseudo code of the algorithm

The provided pseudo code, shown in Fig. 4 finishes the computation in 200 clock cycles.

IV. RESULTS

The implementations have their own advantages and disadvantages. The NPSR offers a simpler design at the cost of less computational power. It can be applied in systems where operations are not frequently executed, for example in integrated power meters [3-4]. Table 2 gives some of achieved results and make comparisons between two architectures. Although, in theory, pipelined architecture should have approximately 18 times larger area in layout, it is not the completely true.

TABLE II
PERFORMANCE METRICS

Metric	Non-pipelined	Pipelined	Ratio
Latency [cycles]	18	18	1
Throughput [ops/cycles]	1/18	1	18
Max. frequency [MHz]	303.03	285.714	0.94
Area [μm^2]	1200	7900	6.58
Dynamic power [W]	0.006	0.072	12
Power efficiency [nJ/operation]	0.356	0.252	0.7

While NPSR operates at slightly higher clock frequency, the PSR achieves 18 times higher processing throughput. The processing efficiency can be computed as a product of frequency maximum and number of operations executed per one clock cycle. The NPSR completes one

operation every 18 cycles, yielding $303.030\text{MHz}/18 = 16.8$ million operations per second (Mops/s). The PSR, in contrast, sustains one operation per cycle at 285.714 MHz, achieving 285.7 Mops/s.

The energy efficiency takes into account both power dissipation and operation rate. The PSR exhibits better energy efficiency despite consuming 12 times higher dynamic power (72 mW vs. 6 mW). The NPSR spends 0.356 nJ per one square root computation. The PSR design, operating at higher throughput, completes each operation in 3.5 ns and consumes 0.252 nJ per operation, representing a 29% improvement in energy efficiency.

V. CONCLUSION

The paper presents three implementations of the iterative radix-2 square root algorithm, synthesized in TSMC 65nm technology, revealing fundamental trade-offs between throughput, area, and energy efficiency. The pipelined architecture achieves superior processing throughput and energy efficiency per operation at the cost of increased area and dynamic power consumption. However, this advantage is possible only when pipeline is fully utilized. In applications with bursty workloads or with long idle periods, the non-pipelined architecture exhibits more energy efficient operation, because of negligible power spent in the idle mode.

ACKNOWLEDGMENT

This work was supported in part by the Ministry of Science, Technological Development and Innovation of the Republic of Serbia under grant numbers 451-03-136/2025-03/200102 and 451-03-137/2025-03/200102.

REFERENCES

- [1] R. P. Villa, J. Rivera, J. J. Raygoza, E. Becerra and S. Ortega, "Digital Circuit Design for the Square Root Computation by Means of Unfolding Techniques", in IEEE Embedded Systems Letters, Vol. 17, No. 1, pp. 62-65, Feb. 2025.
- [2] B.Jovanovic, M. Damjanovic "Digital systems for square root computation", in Proc. of XLVII ETRAN Conference, Herceg Novi, Vol. I, June 8-13, 2003.
- [3] A. Dianov, A. Anuchin and A. Bodrov, "Fast Square Root Calculation without Division for High Performance Control Systems of Power Electronics", in CES Transactions on Electrical Machines and Systems, Vol. 6, No. 2, pp. 145-152, June 2022.
- [4] N. Takagi and K. Takagi, "A VLSI Algorithm for Integer Square-Rooting", Int. Symposium on Intelligent Signal Processing and Communications, Yonago, Japan, pp. 626-629, 2006.