Table-free Seed Generation for Hardware Newton–Raphson Square Root and Inverse Square Root Implementations in IoT Devices

Luis Parrilla, Antonio Lloris, Encarnación Castillo and Antonio García, *Senior Member, IEEE,*

***Abstract*—Restrictions inherent to processors used in Internet of Things (IoT) devices, or the need for area reduction in massively parallel processing systems for artificial intelligence, lead to the search of trade-offs between precision and area in arithmetic operations. In this sense, simple precision enables to save area and to speed up computations when no higher precision is required. On the other hand, iterative methods such as Newton-Raphson's or Goldschmidt's allow to reduce or remove the use of memory tables. In these methods, the selection of the initial value or seed is critical to achieve a reasonable precision using the lowest possible number or iterations. In this paper, new hardware implementations of seed-generators with very low area requirements are proposed for Square Root and Inverse Square Root, while maintaining good precision, high performance and low power consumption. A new design methodology, named SUccessive Approximation Methodology (SUAM), enables to implement seed-generators using only a few logic gates. The presented experimental and implementation results support the validity of the developed seed-generators.**

***Index Terms*—Arithmetic, seed, Newton-Raphson, Square Root, Inverse Square Root, hardware implementation, IoT, Goldschmidt.**

I. Introduction

THE restrictions inherent to processors used in Internet of Things (IoT) devices [1], or the massively parallel processing systems for Artificial Intelligence (AI), are reviving interest in iterative calculation methods operating in simple precision for computing elementary functions, such as square root, exponentiation, or natural logarithm. Simple precision allows to save area and speed up calculations when no higher precision is required, being specially suitable for the majority of IoT applications and computations involved in machine-learning [2]. Moreover, iterative methods allow to reduce or remove the use of memory tables, thus saving the scarce memory resources available in this type of processing systems. One important issue regarding iterative methods is the requirement of a seed as the starting value for the iteration [3]. In the case of square root (SQRT), the most used iterative methods are based on the application of the Newton-Raphson (NR) [4] and the Goldschmidt (GS) [5] [6] numerical procedures. Regarding these, there are two main methods described in the literature for computing square root by means of NR or GS: (1) computing the square root directly using the Newton-Raphson method [7], and (2) computing the inverse square root (ISQRT) by using Newton-Raphson or Goldschmidt iterations with a final multiplication by the operand [5], [8]. This last method is widely used since it avoids division when performing NR iterations [9]. Recently, a modification of NR main iteration has been proposed in [10], which enables a reduction of the maximum relative error in each iteration. Nevertheless, in this article we will use the classical NR formulation in order to easy comparison to other methods and procedures in the literature. Usually, seeds are stored as a set of values in a look-up table (LUT), as in [11] or [12]. In [13], the hardware reuse of a multiplier for generating seeds is proposed. Some works have proposed the use of "magic numbers" combined with simple arithmetic operations as in the "Quake" method [14], [15], or its modifications for improved accuracy [16], [17] (a brief discussion can be found in [18]). In [19] an approach based on the use of low-degree, low-precision polynomials for hardware implementations is presented, and in [20], minimax polynomials are also used for computing seeds, thus providing a table-free hardware implementation of a seed generator focused on achieving high precision at the cost of high area requirements. In this article, we propose a low-area overhead table-free method, in which seeds are computed from the operand by means of simple logical operations, thus optimizing hardware implementations of square root and inverse square root based on iterative approximations. Therefore, this method is specially suitable for embedded systems or IoT devices where restrictions in area and/or memory discourage the use of memory-consuming tables. The rest of the paper is organized as follows: Section II revises the Newton–Raphson method and its application to the computation of square root and inverse square root, Section III and IV are devoted to the computation of optimal tables of seeds for square root and inverse square root, respectively. Section V and VI present the proposed seed generation methods for square root and inverse square root and their corresponding experimental results, respectively. Section VII includes a comparison of the proposed methods to other works in the literature, and Section VIII presents the conclusions of the paper.

II. Newton-Raphson for computing square root and inverse square root

The Newton–Raphson method is a well-known numerical procedure for finding zeros of functions, *i.e.*, for solving:

$$f(x) = 0 \tag{1}$$

L. Parrilla, A. Lloris, E. Castillo and A. García are with the Department of Electronics and Computer Technology, University of Granada, 18071-GRANADA (Spain).
E-mail: lparrilla@ditec.ugr.es

Manuscript received xxx 19, 2021; revised xxx 26, 2021.

TABLE I
NR FOR COMPUTING $\sqrt{5}$ WITH $x_0 = 0$ IN DOUBLE-PRECISION

q	x_q	x_{q+1}	Absolute Error
0	1.0	3.0	0.764
1	3.0	2.3333333333333335	0.0973
2	2.3333333333333335	2.238095238095238	0.00203
3	2.238095238095238	2.2360688956433634	$9.18e-07$
4	2.2360688956433634	2.236067977499978	$1.88e-13$
5	2.236067977499978	2.236067977499978	$< 2.2e-16$

where f is a well-behaved function (continuously differentiable). NR is an iterative method, so if *r* is a root of (1), the procedure starts with an initial value, x_0, close to *r*. Then, successive estimates of *r* are computed as:

$$x_{q+1} = x_q - \frac{f(x_q)}{f'(x_q)} \tag{2}$$

where x_q is the current estimate, and x_{q+1} the next one. Convergence of iterations depends on the behavior of $f(x)$, and the selection of the initial value x_0, also known as "seed". If $f''(x)$ exists and it is continuous near *r*, it can be proved that NR presents locally quadratic convergence [6] to *r*. Thus, the seed is required to be close to *r*, with no other root in the neighborhood.

A. NR for SQRT

In order to compute the square root $r = \sqrt{a}$ using the NR method, we can define $f(x)$ as:

$$f(x) = x^2 - a \tag{3}$$

In this case, (3) has a unique solution in R^+, $f'(x) = 2 \cdot x$ and $f''(x) = 2$, so convergence is guaranteed if $x_0 > 0$. The iteration defined in (2) can be then written as:

$$x_{q+1} = \frac{x_q}{2} + \frac{a}{2x_q} \tag{4}$$

As (4) will not present convergence problems, the seed value has effect only on the number of iterations required for obtaining a given accuracy for the solution. As an example, for computing $\sqrt{5}$ using double-precision floating-point, six iterations are required if $x_0 = 1$, as shown in Table I. If $x_0 = 2$ is used as initial seed, only four iterations are required for achieving double-precision accuracy. Thus, a correct selection of the seed directly impacts on the performance of the SQRT implementation. Other issue of interest is related to the implementation of the iteration described in (4). As it can be observed, basically a division and an addition are required. Division is a complex operation in hardware, specially when double precision is required [21]. The use of ISQRT enables to avoid division, as it will be commented in the next subsection.

B. NR for inverse square root

Inverse Square Root (ISQRT) is interesting by itself, because it is involved in several arithmetic computations [9] and, additionally, can be used for computing SQRT taking into account that:

$$\sqrt{x} = x \cdot \frac{1}{\sqrt{x}} \tag{5}$$

In order to apply the NR method to ISQRT, we have to consider that in this case $f(x) = \frac{1}{x^2} - a$, and the NR iteration will be:

$$x_{q+1} = x_q(1.5 - 0.5ax_q^2) \tag{6}$$

Therefore, only a subtraction and three multiplications are required to compute ISQRT using NR, thus avoiding division. Moreover, the "Quake" method provides a simple seed computation enabling a fast software implementation of ISQRT and SQRT [14]. As in the case of NR applied to SQRT, the value of the seed is critical for performance in the ISQRT computation.

III. OPTIMAL TABLES OF SEEDS

A. Fundamentals

Seed-tables are a fast method for providing initial values required by NR or other iterative methods such as Goldschmidt's [22]. Values to be stored in a seed-table must be carefully selected in order to minimize table size and to optimize error. When the IEEE 754 [23] standard is used for floating point representation, the calculus of SQRT can be split into two operations:

- *Exponent and mantissa adjustment*. If the operand exponent is even, the exponent of the square root will be the half of this exponent. When it is odd, the mantissa will be divided by two (it will be then of the form "0.1xxxx" instead of "1.xxxxx"), and the exponent will be incremented by one (thus resulting in an even number) and then halved.
- *Computation of SQRT of the mantissa.*

Whether the operand is odd or even, the range of the mantissa is [0,2), and this interval is usually uniformly divided in order to simplify addressing the table. As an example, if the four most significant bits of the mantissa are used as table address, a table of 16 values will be obtained. Usually, the half-point of the interval is used as representative of the interval, and the seed is the square root of such value [25] [26]. This is the optimal value when considering monotonic functions, as it is the case of SQRT or ISQRT. Indeed, if $f(x)$ is monotonic in the interval $[a, b]$, the error when using a constant value $f(v)$, $v \in [a, b]$ instead of $f(x)$, $x \in [a, b]$ is:

$$e(x) = |f(x) - f(v)| \tag{7}$$

The expected value of the error in the interval is:

$$E_{ab} = E[e(x)]_{ab} = \int_a^b |f(x) - f(v)|P(x)dx \tag{8}$$

where $P(x)$ is the density of probability of selecting x. If we have the same probability for each $x \in [a,b]$, and $f(x)$ is a monotonic increasing function then we can write:

$$E_{ab} = \frac{1}{b-a}\Big[\int_a^v [f(v)-f(x)]dx + \int_v^b [f(x)-f(v)]dx\Big] = \frac{1}{b-a}\Big[f(v)(v-a) - \int_a^v f(x)dx + \int_v^b f(x)dx - f(v)(b-v)\Big] = \frac{1}{b-a}\Big[f(v)(2v-a-b) - \int_a^v f(x)dx + \int_v^b f(x)dx\Big] \quad (9)$$

Computing the derivative of equation (9) and equaling it to 0 result in:

$$f'(v) = \big[2v-(a+b)\big] = 0 \Rightarrow v = \frac{a+b}{2} \quad (10)$$

Therefore, by evaluating $f(x)$ in the midpoint of the interval the expected error is minimized.

A refinement for the case of using these seeds for NR iterations is proposed in [25], where an optimal evaluation point is obtained for functions of the form $f(x) = x^p$. This evaluation point is given by:

$$v_l = \frac{a^{2^{-l}} + b^{2^{-l}}}{a^{2^{-l}-1} + b^{2^{-l}-1}} \quad (11)$$

and it depends on the number of iterations l to be performed when applying the NR method. Note that for $l = 0$, this evaluation point is $v_0 = \frac{a+b}{2}$, as expected. In the case of $l = 1$, $v_1 = \sqrt{ab}$, thus being the geometric mean. Nevertheless, a final step for generating seed-tables is required: the value $f(v)$ has to be rounded (rather than truncated) accordingly to the precision of the values stored in the table. This leads in practice to obtaining the same final values independently of l. As an example, let us consider the SQRT function and a 16-row, 5-bit precision seed-table. In the following, we will represent the mantissa of the input number as $x = x_0.x_1x_2x_3x_4x_5...$, where x_0 is '0' or '1', and $\sqrt{x}$ as $r_0.r_1r_2r_3r_4r_5....$ Indeed, in the case of even exponents, the mantissa of the number can be represented as $1.x_1x_2x_3x_4x_5...$, and then $r = \sqrt{x}$ will be represented as $1.r_1r_2r_3r_4r_5...$ In the case of odd exponents, we will have to compute the SQRT of $0.x_1x_2x_3x_4x_5..$, which results in $0.r_1r_2r_3r_4r_5....$ Therefore, $r_0 = x_0$, and it is possible to make the integer part implicit thus saving a bit in the seed representation. As an example, Table II shows the seed-table for $n = 4$ and $m = 5$, where n is the number of bits addressing the table, and m is the number of bits representing the mantissa of the seed value. Note that if normal numbers are assumed, the first 2^{n-2} seeds can be removed (all numbers will be represented as "0.1xx" or "1.xxx", depending on the parity of the exponent), thus requiring a table of $2^n - 2^{n-2}$ values. In the case of $n = 4$, a table with 12 seeds will suffice (addresses from "0000" to "0011" have been included in Table II, but with don't-care values "- - - - -").

Using Table II, $\sqrt{0.03125}$ will be computed as it follows:

1) 0.03125 is represented, using a 4-bit truncated mantissa and the exponent, as $1.000 \cdot 2^{-5}$.
2) After exponent and mantissa adjustments, 0.03125 can be represented as $0.100 \cdot 2^{-4}$.
3) Corresponding address in Table II will be "0100".
4) From Table II, the mantissa of the seed will be "11000".
5) The approximate value corresponding to the seed is thus $\sqrt{0.03125}_{seed} = 0.11000 \cdot 2^{-2} = 0.1875$.

TABLE II
SQUARE ROOT SEED-TABLE FOR $n = 4, m = 5$

address	value	address	value
0000	- - - - -	1000	00001
0001	- - - - -	1001	00011
0010	- - - - -	1010	00101
0011	- - - - -	1011	00110
0100	11000	1100	01000
0101	11010	1101	01010
0110	11101	1110	01011
0111	11111	1111	01101

TABLE III
MEAN ABSOLUTE ERROR (MAE), MAXIMUM ABSOLUTE ERROR (MAXAE), MEAN RELATIVE ERROR (MRE), AND MAXIMUM RELATIVE ERROR (MAXRE) FOR DIFFERENT VALUES OF n AND m

n	m	MAE	MAXAE	MRE	MAXRE	$nbits$
4	3	0.0389	0.1250	0.0378	0.1340	36
4	4	0.0226	0.0625	0.0215	0.0646	48
4	5	0.0161	0.0535	0.0157	0.0618	60
4	6	0.0150	0.0448	0.0146	0.0607	72
4	7	0.0148	0.0429	0.0145	0.0607	84
5	3	0.0358	0.1250	0.0342	0.1250	72
5	4	0.0203	0.0625	0.0199	0.0833	96
5	5	0.0111	0.0406	0.0108	0.0513	120
5	6	0.0081	0.0312	0.0079	0.0417	144

The double-precision value of $\sqrt{0.3125}$ is 0.1767766952966369, so the error in this case is < 0.011. Using this table for computing the square root, the Mean Absolute Error (MAE) of all numbers in the range [0.5,2) using single-precision ($2^{23} + 2^{22}$ samples) is 0.0139 and the MAXimum Absolute Error (MAXAE) is 0.0535. Table III shows error figures corresponding to seed-tables built using this method for different values of n and m. Presented values have been obtained computing the square root and the seed of all normal mantissas corresponding to the single-precision IEEE 754 standard [23]. In this table, MRE is the Mean of Relative Error, MAXRE is the Maximum Relative Error, and $nbits$ is the number of bits required for storing the corresponding table of seeds.

Fig. 1 shows the precision (P) for different seed-tables in terms of the number of NR iterations, l, when computing $\sqrt{x}$. Precision is computed as the number of exact bits, *i.e.*, $-log_2(\text{MAXAE})$, where MAXAE is the maximum absolute error for all the single-precision mantissas in the [0.5,2) range. From this figure, the $n = 4, m = 3$ seed-table allows to achieve single precision in 3 NR iterations, while double precision requires 4 NR iterations. In order to achieve single precision in 2 NR iterations and double precision in 3 NR

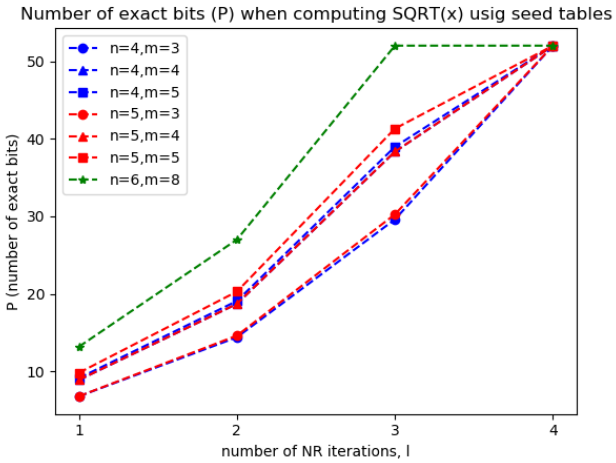


Fig. 1. Precision (P) for different n, m values after l NR iterations.

iterations, the $n = 6, m = 8$ seed-table is required, thus increasing the size of the required table to 384 bits. In systems without high-precision requirements, 20-bit precision can be enough, as provided by a $n = 5, m = 5$ seed-table in 2 NR iterations. It should be noted that increasing m beyond 5 does not improve precision appreciably, so $n = 4, m = 5$ or $n = 5, m = 5$ represent good trade-offs for low-resource implementations.

B. Hardware implementation of tables of Square Root seeds

An immediate approach to hardware implementation of seed-tables consists on using ROM memories. However, this is not an optimal implementation for embedded systems with limited area resources, because of the need of addressing circuits, and the non-minimal representation of information inherent to ROM structures. As it is well known, this can be solved considering a ROM memory as a set of combinational functions that can be minimized using traditional Karnaugh maps, Quine-McCluskey methods, or any other semi-heuristic procedures used in modern design tools [24]. As commented in the introduction, in this paper we focus on direct hardware implementations of NR seeds, thus pursuing to achieve minimal area to obtain SQRT and ISQRT seeds. Therefore, a first approach can be to treat seed-tables as logic functions, minimize them, and implement the resulting minimized functions by using logic gates. Table IV shows implementation results for this solution (Minimized table) on a 45-nm standard-cell process, and also different types of FPGAs. The designs have been optimized by the synthesis tools provided by the device manufacturers. In the case of VLSI implementations, a 45-nm process using the NandGateOpenCellLibrary v1011.01-HR04-20011-01-19 operating at $VDD = 1.25V$ has been used, while it has been implemented with OASYS-RTL V2018.1 from Mentor Graphics. In the case of FPGAs, the Cyclone II family from Intel, and the Spartan 3 and Spartan 6 families from Xilinx have been selected. These are low-cost and low-resource including 4-input LUTs (Spartan 3, Cyclone II) and 6-input LUTs (Spartan 6). Quartus II 13.01 sp1 from Intel and ISE 14.7 from Xilinx were used for these implementations. Power consumption in FPGAs has been estimated assuming 100 million transitions/s at the inputs of the design and, in the case of the 45-nm VLSI process, assuming 500 million transitions/s.

TABLE IV
IMPLEMENTATION RESULTS FOR MINIMIZED LOOK-UP TABLES FOR *SQRT* SEED GENERATION.

n	m	Device/technology	area	delay	power
4	5	Cyclone II FPGA	3 LUT4	3.29ns	49.96mW
4	5	Spartan 3 FPGA	4 LUT4	2.96ns	41.00mW
4	5	Spartan 6 FPGA	3 LUT6	1.73ns	42.00mW
4	5	45 nm NandGate process	8 squm	28.5ps	21.81uW
5	5	Cyclone II FPGA	8 LUT4	3.42ns	58.00mW
5	5	Spartan 3 FPGA	9 LUT4	2.96ns	41.00mW
5	5	Spartan 6 FPGA	3 LUT6	1.73ns	42.00mW
5	5	45nm NandGate process	16 squm	59.2ps	58.38uW

From Table IV, an increase of one unit in n produces a duplication of area requirements in the VLSI process and in 4-input LUT FPGAs. Therefore, we will focus on proposing a new direct hardware implementation close to a $n = 5, m = 5$ ROM-based implementation reducing area requirements. This proposal will be developed in Section V.

IV. Inverse Square Root tables of seeds

As commented in subsection II-B, the Newton Raphson method applied to Inverse Square Root (ISQRT) requires only multiplications, thus enabling the computation of ISQRT and SQRT without using division. Indeed, $\sqrt{x}$ can be computed doing a final product by x, $\sqrt{x} = x \cdot (1/\sqrt{x})$. Thus the number of multiplications needed in this case for computing SQRT will be $2l + 1$, where l is the number of NR iterations, according to equation (6). Note that the ISQRT seeds for NR can be obtained using the same procedure presented in Section III for SQRT, being the results for each of the $2^{23} + 2^{22}$ single-precision values in [0.5,2) those presented in Table V.

Fig. 2 shows the precision achieved when using seed-tables for different number of NR iterations. As it can be observed, the $n = 4, m = 4$ seed-table provides single-precision with 3 iterations, and double-precision in 4. A $n = 8, m = 8$ table is required to achieve double-precision in only 3 iterations (and single-precision in 2).

In the case of using ISQRT for computing SQRT, the corresponding figures for optimal seed-tables are presented in Table VI, which are very similar to those from Table V.

Regarding precision when using ISQRT to compute SQRT with the NR method, Fig. 3 shows that the $n = 4, m = 4$ seed-table provides single-precision (26.35 bits) in 3 NR iterations, and double-precision in 4 iterations. As in the case of computing $1/\sqrt{x}$, a $n = 8, m = 8$ table is required to achieve single-precision (28.3 bits) in 2 iterations, and double precision in only 3 NR iterations.

V. Successive approximation methodology (SUAM) for generating square root seeds

In the case of optimal tables of seeds, the target to be optimized was the absolute value of the error (see equation

TABLE V
MEAN ABSOLUTE ERROR (MAE), MAXIMUM ABSOLUTE ERROR (MAXAE), MEAN RELATIVE ERROR (MRE), AND MAXIMUM RELATIVE ERROR (MAXRE) WHEN COMPUTING $(1/\sqrt{x})_{seed}$ FOR DIFFERENT VALUES OF n AND m

n	m	MAE	MAXAE	MRE	MAXRE	$nbits$
4	3	0.0854	0.3750	0.0873	0.3750	36
4	4	0.0277	0.1017	0.0287	0.0798	48
4	5	0.0189	0.1017	0.0192	0.0719	60
4	6	0.0165	0.1017	0.0160	0.0719	72
4	7	0.0150	0.0861	0.0147	0.0609	84
5	3	0.0660	0.375	0.0678	0.375	72
5	4	0.0272	0.1017	0.0279	0.0719	96
5	5	0.0148	0.0833	0.0148	0.0625	120
5	6	0.0101	0.0548	0.0102	0.0391	144
5	7	0.0079	0.047	0.0078	0.0333	168

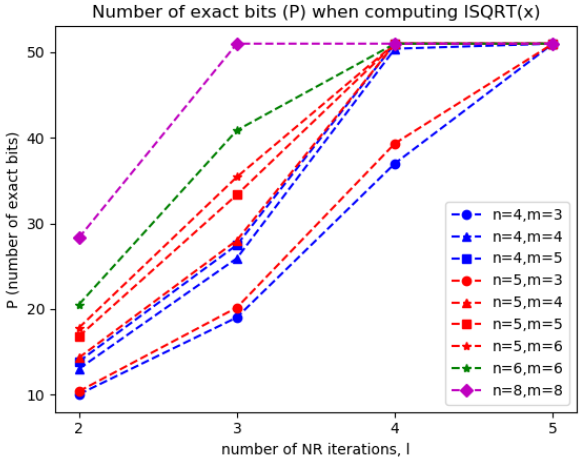


Fig. 2. Precision (P) for different n, m values after l NR iterations when computing $1/\sqrt{x}$.

TABLE VI
MEAN ABSOLUTE ERROR (MAE), MAXIMUM ABSOLUTE ERROR (MAXAE), MEAN RELATIVE ERROR (MRE), AND MAXIMUM RELATIVE ERROR (MAXRE) WHEN COMPUTING $x \cdot (1/\sqrt{x})_{seed}$ FOR DIFFERENT VALUES OF n AND m

n	m	MAE	MAXAE	MRE	MAXRE	$nbits$
4	3	0.0921	0.3750	0.0873	0.3750	36
4	4	0.0308	0.0802	0.0287	0.0798	48
4	5	0.0203	0.0560	0.0192	0.0719	60
4	6	0.0163	0.0509	0.0160	0.0719	72
4	7	0.0150	0.0430	0.0147	0.0609	84
5	3	0.0702	0.375	0.0678	0.375	72
5	4	0.0300	0.0802	0.0279	0.0719	96
5	5	0.0154	0.0467	0.0148	0.0625	120
5	6	0.0108	0.0330	0.0102	0.0391	144
5	7	0.0081	0.0249	0.0078	0.0333	168

(7)). In our proposal, we will optimize the absolute difference between squares of the function:

$$e_s(x) = |f^2(x) - f^2(v)| \tag{12}$$

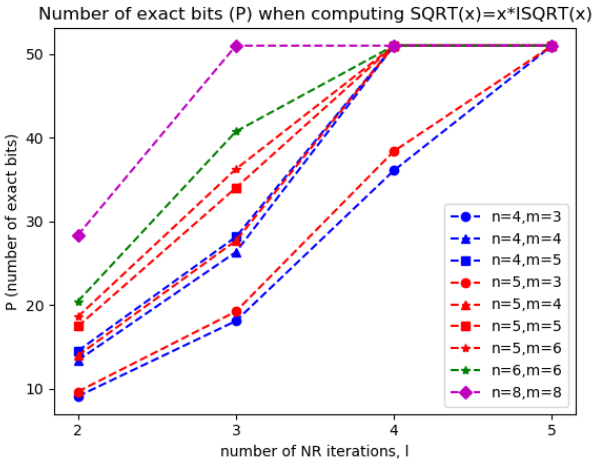


Fig. 3. Precision (P) for different n, m values after l NR iterations when computing $\sqrt{x} = x \cdot (1/\sqrt{x})$.

where x is the point selected for approximating $f(x)$ to $f(v)$ (see Section III). Note that if $f(x) = \sqrt{x}$ we have:

$$x > v \Rightarrow f(x) > f(v) \Rightarrow f^2(x) > f^2(v) \tag{13}$$

Therefore, in the case of $x > v$:

$$e_s(x) = f^2(x) - f^2(v) = (f(x) - f(v)) \cdot (f(x) + f(v)) \tag{14}$$

thus being:

$$e_s(x) = e(x) \cdot (f(x) + f(v)) \tag{15}$$

The same result is obtained when $x < v$. Moreover, if x is a number expressed in the IEEE 754 standard, computation of $\sqrt{x}$ leads to two types of mantissas depending on the exponent of x: $1.x_1x_2x_3x_4x_5...$ and $0.x_1x_2x_3x_4x_5...$, so $f(x)+f(v) \geq 1$ when $f(x)$ is an approximation of $f(v)$. As a consequence:

$$e(x) = \frac{e_s(x)}{f(x) + f(v)} \leq e_s(x) \tag{16}$$

This result shows that optimizing $e_s(x)$ implies optimizing $e(s)$. In the following, we will consider:

$$e'_s(x) = |f^2(x) - x| \tag{17}$$

as the target to optimize, where $f(x)$ is the approximation to $\sqrt{v}$ and $x \simeq v$ is assumed.

In order to simplify the process of obtaining SUAM equations, we will consider two cases corresponding to the two types of mantissas obtained when using the IEEE 754 standard:

1) Case 1: $x = 1.x_1x_2x_3x_4x_5...$: In this case, $r = \sqrt{x}$ will have the form $1.r_1r_2r_3r_4r_5....$ If we want to obtain a seed with only one fractional bit optimizing $e_s(x)$, we can write:

$$e'_s(x) = 1.x_1x_2 - (1.r_1)^2 \geq 0 \tag{18}$$

where $v \simeq x$, and the condition of not having overflow ($r^2 >= 2$) has been imposed. This expression can be rewritten as:

$$e'_s(x) = 1 + x_1 \cdot 2^{-1} + x_2 \cdot 2^{-2} - [(1 + r_1 \cdot 2^{-1})^2] \tag{19}$$

which results, after squaring, in:

$$e'_s(x) = 1 + x_1 \cdot 2^{-1} + x_2 \cdot 2^{-2} - [(1 + r_1) + r_1 \cdot 2^{-2}] \quad (20)$$

From (20), if we want $e'_s(x)$ to be as close to 0 as possible, while not generating a carry to the integer part of r, it is clear that r_1 has to be $r_1 = 0$. If two bits in the fractional part are considered, the square of r will have four fractional bits, thus having:

$$e'_s(x) = 1.x_1x_2x_3x_4 - (1.0r_2)^2 \quad (21)$$

that results in:

$$e'_s(x) = 1 + x_1 \cdot 2^{-1} + x_2 \cdot 2^{-2} + x_3 \cdot 2^{-3} + x_4 \cdot 2^{-4} - [1 + r_2 \cdot 2^{-1} + r_2 \cdot 2^{-4}] \quad (22)$$

From this equation it is immediate that $r_2 = x_1$. Note that r does not depend on x_2, x_3 and x_4, which allows to save three inputs in the corresponding circuit. From this result, and adding a new bit to the representation of the square root, we can write:

$$e'_s(x) = 1.x_1x_2x_3 - (1.0x_1r_3)^2 \geq 0 \quad (23)$$

where equation (15) has been taken into account to limit the x representation to three fractional bits (precision up to the third fractional bit in r). Expanding (23) and removing terms beyond 2^{-3}, we obtain:

$$e'_s(x) = 1 + x_1 \cdot 2^{-1} + x_2 \cdot 2^{-2} + x_3 \cdot 2^{-3} - [1 + x_1^{-1} + r_3 \cdot 2^{-2}] \quad (24)$$

From this last expression it can be stated that $r_3 = x_2$.

In order to get a new bit, r_4, in our approximation, we will consider the expression:

$$e'_s(x) = 1.x_1x_2x_3x_4 - (1.0x_1r_4)^2 \quad (25)$$

Now, expanding the square to the 2^{-4} terms, we have:

$$e'_s(x) = 1 + x_1 \cdot 2^{-1} + x_2 \cdot 2^{-2} + x_3 \cdot 2^{-3} + x_4 \cdot 2^{-4} - [1 + x_1^{-1} + x_2 \cdot 2^{-2} + r_4 \cdot 2^{-3} + (x_1 + x_1x_2) \cdot 2^{-4}] \quad (26)$$

Following again the objective of minimizing $e'_s(x)$ with the condition $e'_s(x) \geq 0$, we can express r_4 as a function of x_1, x_2, x_3 and x_4, obtaining the truth table presented in Table VII. From this table, we can write:

$$r_4 = x_3 \cdot \bar{x}_1 + x_3 \cdot \bar{x}_2 = x_3 \cdot (\bar{x}_1 + \bar{x}_2) \quad (27)$$

Note that r_4 does not depend on x_4 and don't-cares have been used when $e'_s(x)$ was the same for $r_4 = 0$ and $r_4 = 1$. Using the same procedure for obtaining the bit r_5, expressing r_5 as a function of x_1, x_2, x_3 and x_4, and building and minimizing the corresponding truth table, we obtain:

$$r_5 = \bar{x}_1 \cdot x_4 \quad (28)$$

Table VIII summarizes the functions obtained for each bit of the proposed SQRT seed for the case $x_0 = 1$.

TABLE VII
TRUTH TABLE FOR r_4 WHEN $x_0 = 1$

r_4	x_1	x_2	x_3	x_4
0	0	0	0	0
-	0	0	0	1
1	0	0	1	0
1	0	0	1	1
0	0	1	0	0
-	0	1	0	1
1	0	1	1	0
1	0	1	1	1
0	1	0	0	0
-	1	0	0	1
1	1	0	1	0
1	1	0	1	1
0	1	1	0	0
0	1	1	0	1
0	1	1	1	0
0	1	1	1	1

TABLE VIII
PROPOSED FUNCTIONS FOR HARDWARE-GENERATED SQRT SEED WHEN $x_0 = 1$

bit	function
r_0	1
r_1	0
r_2	x_1
r_3	x_2
r_4	$x_3 \cdot (\bar{x}_1 + \bar{x}_2)$
r_5	$\bar{x}_1 \cdot x_4$

TABLE IX
PROPOSED FUNCTIONS FOR HARDWARE-GENERATED SQRT SEED WHEN $x_0 = 0$ AND $x_1 = 1$

bit	function
r_0	0
r_1	1
r_2	x_1
r_3	x_2
r_4	$r_4 = x_3$
r_5	$r_5 = x_4$

2) Case 2: $0.1x_2x_3x_4x_5...$: In the case of numbers with integer part $x_0 = 0$, it will be $x_1 = 1$ because the mantissa with $x_0 = 0$ comes from the adjustment due to an odd exponent. This implies that $r_0 = 0$ and $r_1 = 1$. For the rest of fractional bits, the same reasoning carried out above can be applied, thus obtaining the functions presented in Table IX.

Both cases can be unified including x_0 as an additional

TABLE X
MEAN ABSOLUTE ERROR (MAE), MAXIMUM ABSOLUTE ERROR (MAXAE), MEAN RELATIVE ERROR (MRE), AND MAXIMUM RELATIVE ERROR (MAXRE) FOR THE PROPOSED 5-BIT SUAM-SQRT HARDWARE SEED GENERATOR

MAE	MAXAE	MRE	MAXRE
0.0142	0.0521	0.0132	0.0607

variable, resulting in the following equations:

$$\begin{aligned} r_0 &= x_0 \\ r_1 &= \bar{x}_0 \\ r_2 &= x_1 \\ r_3 &= x_2 \\ r_4 &= x_3 \cdot (\bar{x}_0 + x_0 \cdot \bar{x}_1 + x_0 \cdot \bar{x}_2) \\ r_5 &= x_4 \cdot (x_0 \cdot \bar{x}_1 + \bar{x}_0) \end{aligned} \tag{29}$$

As it can be seen, the proposed seed generator depends on 5 inputs, thus being called 5-bit SUAM-SQRT seed generator.

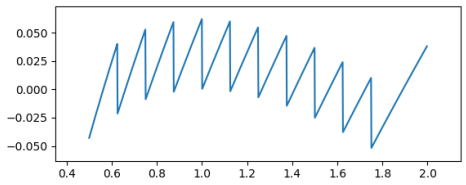
A. Experimental results for SUAM-SQRT seeds

The 5-bit SUAM-SQRT seed generator proposed in equation (29) provides the figures presented in Table X. If we compare these results to optimal seed-tables in Table II, SUAM-SQRT seeds present intermediate results between $n = 4, m = 5$ and $n = 5, m = 5$ rows (SUAM-SQRT provides 6-output values, corresponding to $m = 5$), and very similar to the $n = 4, m = 7$ row.

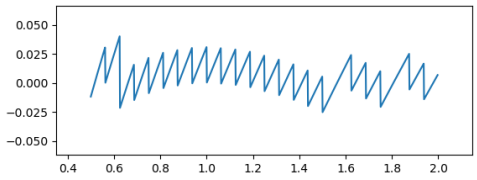
Fig. 4 shows error plots for $n = 4, m = 4$ and $n = 5, m = 5$ seed-tables, and the SUAM-SQRT proposal. Again, it can be seen that SUAM-SQRT presents a compromise between these two seed-tables.

Regarding the results obtained when performing NR iterations starting with SUAM-SQRT seeds, Table XI shows those results, and a comparison to optimal seed-tables with similar characteristics. As it can be observed, $l = 1$ provides almost 10 bits of precision, $l = 2$ iterations nearly 20 bits, $l = 3$ more than 40 bits, and $l = 4$ provides double-precision. Therefore, in the majority of IoT applications, or other such as neural networks where precision is not a priority, one or two NR iterations will suffice.

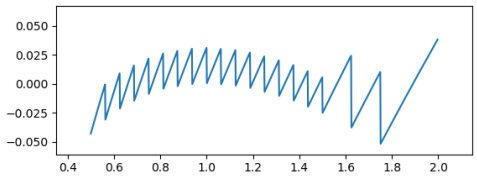
Implementation results for the SUAM-SQRT seed generator are presented in Table XII, and compared to optimal seed-tables with similar performance. As it can be seen, the design proposed in this work presents better figures than all the other alternatives. Concretely, when compared to the seed-table with $n = 5, m = 5$, which presents similar precision, our proposal requires only a 37.5% of the area required by that seed-table, and presents a 39.5% of the seed-table delay when implemented on the 45-nm NandGate process, while precision is only slightly worse (between 0 and 2.7 %) for $l = 1, 2, 3, 4$ NR iterations, as shown in Table XI and Table XIII. Our design is also clearly better in terms of power consumption in the 45-nm NandGate implementation, requiring only 14.3% of the power of the other alternatives. Regarding FPGA technology, SUAM-SQRT only needs from 37.5% to 66.7% of the area required by the $n = 4, m = 5$ seed-table.



(a) $n = 4, m = 4$ seed-table error in [0.5,2) range



(b) $n = 5, m = 5$ seed-table error in [0.5,2) range



(c) SUAM-SQRT error in [0.5,2) range

Fig. 4. Error plots for high efficiency seed-tables and the proposed hardware seed generator

TABLE XI
MEAN ABSOLUTE ERROR (MAE), MAXIMUM ABSOLUTE ERROR (MAXAE), MEAN RELATIVE ERROR (MRE), MAXIMUM RELATIVE ERROR (MAXRE), AND PRECISION (P) OF SUAM-SQRT SEED GENERATOR FOR DIFFERENT VALUES OF l(NUMBER OF ITERATIONS OF NR METHOD)

Design	l	MAE	MAXAE	MRE	MAXRE	P
$n = 5, m = 4$	1	2.82e-4	2.09e-3	2.88e-4	2.23e-3	8.91
$n = 5, m = 5$	1	9.59e-5	1.09e-3	9.96e-5	1.39e-3	9.83
SUAM-SQRT	1	1.43e-4	1.23e-3	1.37e-4	1.73e-3	9.67
$n = 5, m = 4$	2	1.22e-7	2.31e-6	1.32e-7	2.48e-6	18.72
$n = 5, m = 5$	2	1.58e-8	7.60e-7	1.83e-8	9.62e-7	20.30
SUAM-SQRT	2	2.33e-8	1.06e-6	2.87e-8	1.50e-6	19.84
$n = 5, m = 4$	3	6.51e-14	2.87e-12	7.10e-14	3.08e-12	38.34
$n = 5, m = 5$	3	2.20e-15	3.66e-13	2.75e-15	4.63e-13	41.31
SUAM-SQRT	3	3.87e-15	7.97e-13	5.86e-15	1.13e-13	40.19
$n = 4, m = 7$	4	4.62e-17	2.22e-16	4.15e-17	2.22e-16	52.00
$n = 5, m = 4$	4	4.62e-17	2.22e-16	4.15e-17	2.22e-16	52.00
$n = 5, m = 5$	4	4.62e-17	2.22e-16	4.15e-17	2.22e-16	52.00
SUAM-SQRT	4	3.97e-17	2.22e-16	4.15e-17	2.22e-16	52.00

VI. SUCCESSIVE APPROXIMATION METHODOLOGY FOR GENERATING INVERSE SQUARE ROOT SEEDS (ISQRT-SUAM)

In the case of ISQRT, the methodology followed to obtain a seed has to be slightly modified. Indeed, we have:

$$f(x)^2 = \left(\frac{1}{\sqrt{x}}\right)^2 = \frac{1}{x} \tag{30}$$

and $1/x$ presents difficulties to be expressed in terms of the binary digits of x. However, if we multiply (30) by x, we

TABLE XII
IMPLEMENTATION RESULTS OF SUAM-SQRT COMPARED TO SIMILAR PRECISION SEED-TABLES TECHNOLOGIES

Design	Device/technology	area	delay	power
$n=4, m=5$ Table	Cyclone II FPGA	3 LUT4	3.29ns	49.96mW
$n=4, m=5$ Table	Spartan 3 FPGA	4 LUT4	2.96ns	41.00mW
$n=4, m=5$ Table	Spartan 6 FPGA	3 LUT6	1.73ns	42.00mW
$n=4, m=5$ Table	45 nm process	8 squm	28.5ps	21.81uW
$n=4, m=6$ Table	Cyclone II FPGA	4 LUT4	2.86ns	50.83mW
$n=4, m=6$ Table	Spartan 3 FPGA	5 LUT4	2.96ns	41.00mW
$n=4, m=6$ Table	Spartan 6 FPGA	3 LUT6	1.73ns	43.00mW
$n=4, m=6$ Table	45 nm process	11 squm	62.7ps	37.61uW
$n=4, m=7$ Table	Cyclone II FPGA	5 LUT4	2.83ns	52.11mW
$n=4, m=7$ Table	Spartan 3 FPGA	6 LUT4	2.96ns	41.00mW
$n=4, m=7$ Table	Spartan 6 FPGA	4 LUT6	1.73ns	44.00mW
$n=4, m=7$ Table	45 nm process	16 squm	40.6ps	45.31uW
$n=5, m=5$ Table	Cyclone II FPGA	8 LUT4	3.42ns	58.00mW
$n=5, m=5$ Table	Spartan 3 FPGA	9 LUT4	2.96ns	41.00mW
$n=5, m=5$ Table	Spartan 6 FPGA	3 LUT6	1.73ns	42.00mW
$n=5, m=5$ Table	45nm process	16 squm	59.2ps	58.38uW
5-bit SUAM-SQRT	Cyclone II FPGA	3 LUT4	2.82ns	49.84mW
5-bit SUAM-SQRT	Spartan 3 FPGA	2 LUT4	2.96ns	41.00mW
5-bit SUAM-SQRT	Spartan 6 FPGA	2 LUT6	1.73ns	42.00mW
5-bit SUAM-SQRT	45nm process	6 squm	23.4ps	8.32uW

TABLE XIII
IMPLEMENTATION RESULTS OF 5-BIT SUAM-SQRT COMPARED TO $n=5, m=5$ SEED-TABLE (REFERENCE DESIGN, 100%)

Device/technology	area (%)	delay (%)	power (%)	l	P
Cyclone II FPGA	37.5%	82.5%	85.9%	1	98.4%
				2	97.7%
				3	97.3%
Spartan 3 FPGA	22.2%	100%	100%	1	98.4%
				2	97.7%
				3	97.3%
Spartan 6 FPGA	66.7%	100%	100%	1	98.4%
				2	97.7%
				3	97.3%
45nm process	37.5%	39.5%	14.3%	1	98.4%
				2	97.7%
				3	97.3%

obtain:

$$\left(\frac{1}{\sqrt{x}}\right)^2 \cdot x = 1 \tag{31}$$

In terms of the error, it is equivalent to transform (12) in:

$$e_s(x) \cdot v = \left|1 - v \cdot f^2(x)\right| \doteq e_{one}(x) \tag{32}$$

thus having from (15) that:

$$e(x) = \frac{1}{v \cdot [f(x) + f(v)]} \cdot e_{one}(x) \tag{33}$$

Taking into account that v, x are in $[0.5, 2)$ and $x \simeq v$, it is easy to check that $e(x) \leq e_{one}(x)$. Therefore, in this case we will optimize:

$$e'_{one}(x) = \left|1 - x \cdot \left(\frac{1}{\sqrt{x}}\right)^2\right| \tag{34}$$

From (34) it is possible to extract conditions for the digits of $s = (1/\sqrt{x}) = s_0.s_1s_2...s_{n-1}$ from $x = x_0.x_1x_2x_3...x_{n-1}$.

TABLE XIV
TRUTH TABLE DERIVATION FOR s_2 WHEN $x_0 = 1$

$x_1x_2x_3x_4$	$e'_{one}(x)\big\vert_{s_2=0}$	$e'one(x)\big\vert_{s_2=1}$	s_2
0 0 0 0	0.111000	0.011100	1
0 0 0 1	0.101111	0.011010	1
0 0 1 0	0.101110	0.011000	1
0 0 1 1	0.101101	0.010110	1
0 1 0 0	0.101100	0.010011	1
0 1 0 1	0.101011	0.010001	1
0 1 1 0	0.101010	0.001111	1
0 1 1 1	0.101001	0.001101	1
1 0 0 0	0.101000	0.001010	1
1 0 0 1	0.100111	0.001000	1
1 0 1 0	0.100110	0.000110	1
1 0 1 1	0.100101	0.000100	1
1 1 0 0	0.100100	0.000001	1
1 1 0 1	0.100011	0.000001	1
1 1 1 0	0.100010	0.000011	1
1 1 1 1	0.100001	0.000101	1

Let us show an example for the case of $x_0 = 1$ $(x \geq 1)$. In this case, it is clear that $\sqrt{x} \geq 1$ and $s = (1/\sqrt{x}) \leq 1$. This implies that $s_0 = 0$ (except in the only case of $x = 1$). If we do a first approach with one fractional bit for the seed, we will have:

$$e'_{one}(x) = \left|1 - 1.x_1x_2 \cdot (0.s_1)^2\right| \tag{35}$$

or:

$$e'_{one}(x) = \left|1 - s_1 \cdot 2^{-2} - s_1x_1 \cdot 2^{-3} - s_1x_2 \cdot 2^{-4}\right| \tag{36}$$

In this case, it is clear that necessarily $s_1 = 1$, independently of x_1 and x_2, in order to generate the closest value to "1.0000".

Now, to obtain the bit s_2 for the case of $x_0 = 1$, we will have the following expression:

$$e'_{one}(x) = \left|1 - 1.x_1x_2x_3x_4 \cdot (0.1s_2)^2\right| \tag{37}$$

Expanding the square and considering terms down to 2^{-6} (for to take into account inputs until x_4), we obtain:

$$\begin{aligned} e'_{one}(x) = \Big|1 - \big\{&(1+s_2)\cdot 2^{-2} + (1+s_2)x_1 \cdot 2^{-3} + \\ &[s_2 + (1+s_2)x_2]\cdot 2^{-4} + [x_1s_2 + (1+s_2)x_3]\cdot 2^{-5} + \\ &[s_2x_2 + (1+s_2)x_4)]\cdot 2^{-6}\big\}\Big| \end{aligned} \tag{38}$$

From (38) we have to select the value of s_2 generating the lowest $e'_{one}(x)$ for each $1.x_1x_2x_3x_4$ input. For easing this task, (38) can be split into two equations, one for $s_2 = 0$, and other for $s_2 = 1$, as follows:

$$\begin{aligned} e'_{one}(x)\big\vert_{s_2=0} = \Big|1 - \big\{&2^{-2} + x_1 \cdot 2^{-3} + x_2 \cdot 2^{-4} \\ &+ x_3 \cdot 2^{-5} + x_4 \cdot 2^{-6}\big\}\Big| \end{aligned} \tag{39}$$

$$\begin{aligned} e'_{one}(x)\big\vert_{s_2=1} = \Big|1 - \big\{&2^{-1} + x_1 \cdot 2^{-2} + x_2 \cdot 2^{-3} \\ &+ (1+x_3)\cdot 2^{-4} + (x_1+x_4)\cdot 2^{-5} + x_2 \cdot 2^{-6}\big\}\Big| \end{aligned} \tag{40}$$

Table XIV shows the $e'_{one}(x)$ values obtained in each case, the selected s_2 value, and the corresponding truth table. As it can be seen, in all cases $s_2 = 1$ provides the best approach to the final value "1.000000". Repeating this process for s_3 and s_4, and making the same study for the case $x_0 = 0$, Table XV has been derived, thus obtaining a proposal for an efficient

TABLE XV
PROPOSED FUNCTIONS FOR HARDWARE-GENERATED 5-BIT SUAM-ISQRT SEED

bit	function when $x_0 = 0$	function when $x_0 = 1$
s_0	1	0
s_1	0	1
s_2	$\bar{x}_2\bar{x}_3 + \bar{x}_2\bar{x}_4$	1
s_3	$x_2\bar{x}_3 + \bar{x}_2x_3x_4 + \bar{x}_3\bar{x}_4$	$\bar{x}_1(\bar{x}_2 + \bar{x}_3)$
s_4	$\bar{x}_2x_4 + x_2x_3\bar{x}_4$	$\bar{x}_2\bar{x}_3 + \bar{x}_1x_3$

hardware implementation for the generation of ISQRT seeds. Equations in this table have been unified as:

$$\begin{aligned} s_0 &= \bar{x}_0 \\ s_1 &= x_0 \\ s_2 &= x_0 + \bar{x}_0(\bar{x}_2\bar{x}_3 + \bar{x}_2\bar{x}_4) \\ s_3 &= \bar{x}_0(x_2\bar{x}_3 + \bar{x}_2x_3x_4 + \bar{x}_3\bar{x}_4) + x_0\bar{x}_1(\bar{x}_2 + \bar{x}_3) \\ s_4 &= \bar{x}_0(\bar{x}_2x_4 + x_2x_3\bar{x}_4) + x_0(\bar{x}_2\bar{x}_3 + \bar{x}_1x_3) \end{aligned} \tag{41}$$

Note that we have used 5 inputs ($x_0.x_1x_2x_3x_4$) to perform the different products, so the proposed hardware seed-generator has been called 5-bit SUAM-ISQRT.

Nevertheless, from Fig. 2 and Fig. 3, a seed obtained from a 4-bit input requires the same number of NR iterations as the $n = 5$ case to achieve single-precision (3 iterations for $m \geq 4$) and double-precision (4 iterations for $m \geq 4$). Therefore, we will also build a 4-bit SUAM-ISQRT hardware seed generator, which will lead to simpler equations. Concretely, equations corresponding to the so called 4-bit_opt SUAM-ISQRT seed generator are the following:

$$\begin{aligned} s_0 &= \bar{x}_0 \\ s_1 &= x_0 \\ s_2 &= x_0 + \bar{x}_0\bar{x}_2 \\ s_3 &= \bar{x}_0\bar{x}_3 + x_0\bar{x}_1(\bar{x}_2 + \bar{x}_3) \\ s_4 &= x_0(\bar{x}_1\bar{x}_2 + x_1\bar{x}_2\bar{x}_3 + \bar{x}_1x_2x_3) \end{aligned} \tag{42}$$

A. Experimental results for SUAM-ISQRT

Table XVI shows error figures for the two proposed ISQRT hardware-seed generators, when used for computing $1/\sqrt{x}$ and $\sqrt{x} = x \cdot (1/\sqrt{x})$. From this table, 5-bit SUAM-ISQRT provides results close to the $n = 5, m = 5$ seed-table, while 4-bit_opt presents slightly better results than the $n = 4, m = 4$ table. When SUAM-ISQRT seed generators are used in NR iterations, the results when computing ISQRT are presented in Table XVII. Note that both 4-bit_opt SUAM-ISQRT, and 5-bit SUAM-ISQRT provide single-precision with 3 NR iterations, while 4 iterations are required for double-precision with 5-bit SUAM-ISQRT and 5 iterations in the case of 4-bit_opt SUAM-ISQRT.

In the case of computing SQRT from ISQRT, the results are presented in Table XVIII, where 3 iterations are required again for single-precision, 4 iterations for double-precision using 5-bit SUAM-ISQRT, and 5 iterations for double-precision by means of 4-bit_opt SUAM-ISQRT. Note that one more iteration is required in both cases when compared to the direct computation of NR-SQRT but, in return, no division operation is performed.

TABLE XVI
MEAN ABSOLUTE ERROR (MAE), MAXIMUM ABSOLUTE ERROR (MAXAE), MEAN RELATIVE ERROR (MRE), AND MAXIMUM RELATIVE ERROR (MAXRE) FOR THE PROPOSED USAM-ISQRT HARDWARE SEED GENERATORS

design	operation	MAE	MAXAE	MRE	MAXRE
4-bit_opt	ISQRT	0.0257	0.1101	0.0266	0.087
4-bit_opt	SQRT	0.0287	0.0858	0.0266	0.087
5-bit	ISQRT	0.0195	0.0625	0.0213	0.0625
5-bit	SQRT	0.0242	0.0858	0.0213	0.0625

TABLE XVII
MEAN ABSOLUTE ERROR (MAE), MAXIMUM ABSOLUTE ERROR (MAXAE), MEAN RELATIVE ERROR (MRE), MAXIMUM RELATIVE ERROR (MAXRE), MEAN PRECISION (MP), AND MINIMUM PRECISION (MINP) FOR DIFFERENT VALUES OF n, m AND l(NUMBER OF ITERATIONS OF NR METHOD) WHEN COMPUTING ISQRT

design	l	MEA	MAXEA	MRE	MAXRE	P
4-bit_opt	2	1.03e-5	2.58e-4	9.62e-6	2.04e-4	11.92
4-bit_opt	3	1.11e-9	7.91e-8	9.49e-10	6.26e-8	23.59
4-bit_opt	4	9.28e-17	7.77e-15	8.87e-17	6.14e-15	46.87
5-bit	2	3.21e-6	4.93e-5	3.72e-6	4.93e-5	14.31
5-bit	3	9.40e-10	3.64e-9	1.12e-10	3.64e-9	28.03
5-bit	4	5.48e-17	4.44e-16	5.71e-17	3.55e-16	51.00

TABLE XVIII
MEAN ABSOLUTE ERROR (MAE), MAXIMUM ABSOLUTE ERROR (MAXAE), MEAN RELATIVE ERROR (MRE), MAXIMUM RELATIVE ERROR (MAXRE), MEAN PRECISION (MP), AND MINIMUM PRECISION (MINP) FOR DIFFERENT VALUES OF n, m AND l(NUMBER OF ITERATIONS OF NR METHOD) WHEN COMPUTING SQRT FROM ISQRT

design	l	MEA	MAXEA	MRE	MAXRE	P
4-bit_opt	2	9.36e-6	1.61e-4	9.62e-6	2.04e-4	12.60
4-bit_opt	3	8.25e-10	4.95e-8	9.49e-10	6.26e-8	24.27
4-bit_opt	4	8.19e-17	4.77e-15	8.36e-17	6.04e-15	47.57
5-bit	2	4.43e-6	6.71e-5	3.72e-6	4.93e-5	13.86
5-bit	3	1.36e-10	4.78e-9	1.12e-10	3.64e-9	27.64
5-bit	4	5.64e-17	4.44e-16	5.22e-17	3.41e-16	51.00

Regarding implementation results, Table XIX shows area, delay and power figures for the two proposed SUAM-ISQRT implementations when compared to similar precision seed-tables. Note that SUAM-ISQRT provides better implementation figures than the corresponding seed-tables, especially when implemented on the 45-nm process, while maintaining the same number of required NR iterations for achieving single-precision and double-precision, as established in the IEEE 754 standard.

VII. COMPARISON TO OTHER SEED-GENERATORS

Regarding other methods for seed generation in the literature, the majority of them are based on the use of tables, as in [22] or [11]. Besides, they are oriented to using ISQRT for SQRT computation as they take advantage of using only multiplications when applying NR. To the best of our knowledge, only the so called "Quake method" [14] and works in [15] and [20] can be considered as seed-generators. The Quake method

TABLE XIX
AREA, DELAY AND POWER FIGURES FOR OPTIMAL TABLES AND SUAM PROPOSED DESIGN ON DIFFERENT TECHNOLOGIES.

Design	Device/technology	area	delay	power
$n=4, m=4$ Table	Cyclone II FPGA	3 LUT4	2.94ns	51.49mW
$n=4, m=4$ Table	Spartan 3 FPGA	3 LUT4	2.96ns	41.00mW
$n=4, m=4$ Table	Spartan 6 FPGA	3 LUT6	1.73ns	41.00mW
$n=4, m=4$ Table	45 nm process	9 squm	39.0ps	30.13uW
$n=4, m=5$ Table	Cyclone II FPGA	4 LUT4	3.07ns	51.02mW
$n=4, m=5$ Table	Spartan 3 FPGA	5 LUT4	2.96ns	41.00mW
$n=4, m=5$ Table	Spartan 6 FPGA	3 LUT6	1.73ns	42.00mW
$n=4, m=5$ Table	45 nm process	12 squm	37.1ps	30.25uW
$n=5, m=4$ Table	Cyclone II FPGA	6 LUT4	3.11ns	52.69mW
$n=5, m=4$ Table	Spartan 3 FPGA	5 LUT4	2.96ns	41.00mW
$n=5, m=4$ Table	Spartan 6 FPGA	3 LUT6	1.73ns	42.00mW
$n=5, m=4$ Table	45nm process	15 squm	47.6ps	40.26uW
$n=5, m=5$ Table	Cyclone II FPGA	9 LUT4	3.60ns	57.35mW
$n=5, m=5$ Table	Spartan 3 FPGA	8 LUT4	2.96ns	41.00mW
$n=5, m=5$ Table	Spartan 6 FPGA	3 LUT6	1.73ns	42.00mW
$n=5, m=5$ Table	45nm process	21 squm	70.1ps	69.46uW
4-bit_opt SUAM-ISQRT	Cyclone II FPGA	3 LUT4	3.13ns	50.91mW
4-bit_opt SUAM-ISQRT	Spartan 3 FPGA	3 LUT4	2.96ns	41.00mW
4-bit_opt SUAM-ISQRT	Spartan 6 FPGA	3 LUT6	1.73ns	41.00mW
4-bit_opt SUAM-ISQRT	45nm process	5.9 sqm	26.3ps	12.66 uW
5-bit SUAM-ISQRT	Cyclone II FPGA	5 LUT4	3.27ns	52.76mW
5-bit SUAM-ISQRT	Spartan 3 FPGA	5 LUT4	2.96ns	41.00mW
5-bit SUAM-ISQRT	Spartan 6 FPGA	3 LUT6	1.73ns	42.00mW
5-bit SUAM-ISQRT	45nm process	11 squm	40.0ps	32.23uW

TABLE XX
MEAN ABSOLUTE ERROR (MAE), MAXIMUM ABSOLUTE ERROR (MAXAE), MEAN RELATIVE ERROR (MRE), MAXIMUM RELATIVE ERROR (MAXRE), MEAN PRECISION (MP), AND MINIMUM PRECISION (MINP) FOR DIFFERENT VALUES OF n, m AND l(NUMBER OF ITERATIONS OF NR METHOD) WHEN COMPUTING ISQRT

design	l	MEA	MAXEA	MRE	MAXRE	P
quake (sqrt)	0	0.0218	0.0416	0.023	0.034	4.59
quake (sqrt)	1	8.81e-4	2.04e-3	9.30e-4	1.75e-3	8.93
quake (sqrt)	2	1.67e-6	4.93e-6	1.78e-6	4.59e-6	17.63
quake (sqrt)	3	9.57e-9	2.88e-8	1.04e-8	2.87e-8	25.05
quake (isqrt)	0	0.0317	0.448	0.023	0.034	1.16
quake (isqrt)	1	1.28e-3	1.93e-2	9.30e-4	1.75e-3	5.69
quake (isqrt)	2	2.46e-6	4.57e-6	1.78e-6	4.61e-6	14.42
quake (isqrt)	3	1.54e-8	5.31e-7	1.04e-8	2.87e-8	20.84

TABLE XXI
COMPARISON OF PRECISION, AREA RESOURCES AND DELAY TO OTHER METHODS IN SPARTAN 3 FGPA DEVICES AND 45-NM PROCESS

design	P	Area (FPGA)	Delay (FPGA)	Area (ASIC)	Delay (ASIC)
$n=5, m=5$ opt. table	6.1	8 LUT4	2.96 ns	21sqm	70.1ps
4-bit_opt SUAM-ISQRT	5.3	3 LUT4	2.96ns	5.9sqm	26.3ps
5-bit SUAM-ISQRT	5.7	5 LUT4	2.96ns	11sqm	40.0ps
[20] with $n=2, g=1$	6.1	12 LUT4	5.9 ns	-	-
[15]	2.3	17 LUT4	2.96ns	-	-

is intended for software, thus it is not competitive for hardware implementations (it requires a 32-bit subtractor), but we have included it for comparing its precision with optimal tables. In this sense, Table XX shows that the "Quake" method provides poor results (worse than the $n=4, m=3$ table) except in the case of computing SQRT using 2 NR iterations. In the rest of cases, seed-tables and the SUAM-ISQRT hardware generator provide better results. In any case, for achieving single-precision, $l=3$ is required. Note that this method cannot provide double-precision because it is intended for software implementations on 32-bit numbers. In the case of the other two works, [20] provides an approach based on minimax polynomial interpolation into the intervals. It is focused on achieving high precision, thus requiring also large area resources. Nevertheless, there are parameter values that lead to reduced area, such as $n=2, g=1$, providing a precision similar to the optimal table $n=5, m=5$, as shown in Table XXI. As it is shown in this table, where implementation results have been obtained on Spartan 3 FPGAs for comparison purposes, and also on a 45-nm ASIC process when possible, the best trade-off between precision and area resources are provided by SUAM-ISQRT implementations. The proposal in [15] provides poor precision while requiring large area resources.

VIII. CONCLUSION

In this paper a new methodology for building hardware optimized seed-generators for square root and inverse square root requiring low area resources has been presented. As a result, three low-cost implementations have been carried out. The first one, SUAM-SQRT, enables the generation of seeds for computing square root using the Newton-Raphson method achieving 20-bit precision in 2 iterations, while requiring 37.4% less area in ASIC implementations than equivalent optimized seed-tables, with only a 2.7% deterioration in precision. Moreover, SUAM-SQRT provides better delay and power consumption figures, showing reductions of 61.5% and 85.7%, respectively. In the case of Inverse Square Root, two hardware implementations have been proposed: 4-bit_opt SUAM-ISQRT and 5-bit SUAM-ISQRT. 4-bit_opt SUAM-ISQRT allows to achieve single precision when computing ISQRT and SQRT in 3 NR iterations, while requiring around 52% less area than equivalent optimized seed-tables over a 45-nm process. Regarding performance and power consumption, similar improvements have been achieved. Comparison to other seed-generators in the literature reinforces the superiority of the proposed hardware implementation for embedded systems with limited area resources and not requiring high precision, as in IoT applications or massively parallel processing units used in AI.

ACKNOWLEDGMENT

This work was partially financed by the Consejería de Economía y Conocimiento de la Junta de Andalucía (Spain) and the European Regional Development Funds (ERDF) under project B-TIC-588-UGR20.

REFERENCES

[1] S. Balaji, K. Nathani, R. SanthakumarSANTHAKUMAR, "IoT technology, applications and challenges: a contemporary survey." Wireless personal communications, vol. 108, no 1, p. 363–388. 2019

[2] M. Abadi, *et al*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283. 2016.

[3] S. D. Conte and C. De Boor, *Elementary numerical analysis: an algorithmic approach*, 3rd ed. Philadelphia, USA: SIAM, 2018.

[4] T. J. Ypma, "Historical development of the Newton–Raphson method," *SIAM review*, vol.37, no. 4, pp. 531–551. 1995.

[5] P. Markstein, "Software division and square root using Goldschmidt's algorithms". In *Proceedings of the 6th Conference on Real Numbers and Computers (RNC'6)*, vol. 123, pp. 146–157. November, 2004.

[6] J.F. Bonnans, J.C. Gilbert, C. Lemaréchal and C.A. Sagastizábal, *Numerical optimization: theoretical and practical aspects*. Springer Science & Business Media. 2006.

[7] A. J. Thakkar and A. Ejnioui (2006). "Design and implementation of double precision floating point division and square root on FPGAs," in *2006 IEEE Aerospace Conference*, pp. 1–7. 2006.

[8] A. H. Karp, P. Markstein, and D. Brzezinski. *U.S. Patent No. 5,515,308*. Washington, DC: U.S. Patent and Trademark Office. 1996.

[9] M. D. Ercegovac, T. Lang, J. M. Muller and A. Tisserand, "Reciprocation, square root, inverse square root, and some elementary functions using small multipliers." *IEEE Transactions on computers*, vol. 49, no. 7, pp. 628–637. 2000.

[10] C. J. Walczyk, L. V. Moroz, and J. L. Cieśliński, "Improving the Accuracy of the Fast Inverse Square Root by Modifying Newton–Raphson Corrections," *Entropy*, vol. 23, no. 1, p. 86, 2021.

[11] T. J. Kwon and J. Draper, "Floating-point division and square root implementation using a Taylor-series expansion algorithm with reduced look-up tables," in *2008 51st Midwest Symposium on Circuits and Systems*, Knoxville, TN, 2008, pp. 954-957.

[12] P. Soderquist and M. Leeser, "An area/performance comparison of subtractive and multiplicative divide/square root implementations," in *Proceedings of the 12th Symposium on Computer Arithmetic*, pp. 132–139. 1995.

[13] E. M. Schwarz and M. J. Flynn, "Hardware starting approximation method and its application to the square root operation," in *IEEE Transactions on Computers*, vol. 45, no. 12, pp. 1356-1369, 1996.

[14] L. V. Moroz, C. J. Walczyk, A. Hrynchyshyn, V. Holimath, and J. L. Cieśliński. "Fast calculation of inverse square root with the use of magic constant-analytical approach," *Applied Mathematics and Computation*, vol. 316, pp. 245-255. 2018.

[15] A. Hasnat, T. Bhattacharyya, A. Dey, S. Halder and D. Bhattacharjee, "A fast FPGA based architecture for computation of square root and Inverse Square Root," in *IEEE 2017 Devices for Integrated Circuit Conference (DevIC)*, Kalyani, pp. 383–387. 2017.

[16] F. Blinn, *Jim Blinn's Corner: Notation, Notation, Notation* (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling). San Mateo, CA, USA: Morgan Kaufmann, 2003.

[17] C. J. Walczyk, L. V. Moroz, and J. L. Cieśliński, "A modification of the fast inverse square root algorithm," Computation, vol. 7, no. 3, p. 41, 2019.

[18] J. M. Muller, "Elementary Functions and Approximate Computing," in Proceedings of the IEEE, vol. 108, no. 12, pp. 2136-2149, 2020.

[19] R. Michard, A. Tisserand and N. Veyrat-Charvillon, "Small FPGA polynomial approximations with 3-bit coefficients and low-precision estimations of the powers of x," *2005 IEEE International Conference on Application-Specific Systems, Architecture Processors (ASAP'05)*, 2005, pp. 334-339.

[20] M. Ercegovac, J. M. Muller and A. Tisserand. "Simple Seed Architectures for Reciprocal and Square Root Reciprocal," *RR-5720, INRIA*, pp.25. 2005.

[21] B. Parhami. *Computer arithmetic*. Oxford university press, 2010.

[22] J. A. Piñeiro, J. D. Bruguera. "High-speed double-precision computation of reciprocal, division, square root, and inverse square root". IEEE Transactions on Computers, vol. 51, no 12, p. 1377–1388, 2002.

[23] IEEE, *Standard for Binary Floating Point Arithmetic, IEEE754*, pp. 1-58. 2008.

[24] S. Gayathri and T. C. Taranath, "RTL synthesis of case study using design compiler," 2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT), Mysuru, pp. 1-7. 2017.

[25] P. Kornerup, J. M. Muller. "Choosing starting values for certain Newton–Raphson iterations". Theoretical computer science, vol. 351, no 1, p. 101-110, 2006.

[26] D. DasSarma and D. W. Matula, "Measuring the accuracy of ROM reciprocal tables," in IEEE Transactions on Computers, vol. 43, no. 8, pp. 932-940, Aug. 1994.

Luis Parrilla received the M.Sc. degree in Physics (majoring in electronics), the M.A.Sc. degree in Electronic Engineering, and the Ph.D. degree in Physics from the University of Granada (Granada, Spain) in 1993, 1995, and 1997, respectively. In 1995 he joined the Department of Electronics and Computer Technology at the University of Granada where he serves as a Professor since 2000. He is author of more than 70 technical papers in international journals and conferences and he serves as reviewer and guest editor for several journals. His current research interests include the protection of IP cores on VLSI and FPGA based systems, the development of high-performance arithmetic and algebraic circuits for IoT and cryptographic applications, and the design of specific architectures for cryptographic processors and biosignal processing.

Antonio Lloris received the M.Sc. and Ph.D. degrees from the Universidad Complutense, Madrid, Spain. He has been a Full Professor at the University of Granada, Granada, Spain. He was with the Centro de Investigaciones Técnicas de Guipúzcoa, San Sebastián, Spain, as a Researcher and as a Lecturer with the Escuela Técnica Superior de Ingenieros Industriales de San Sebastián, Guipúzcoa, Spain. He was with the University of Málaga, Málaga, Spain, and the University of Murcia, Murcia, Spain. His research interests include multiple-value logic, arithmetic and algebraic circuits, testing of digital circuits, and signal processing using the residue number system.

Encarnación Castillo received the M.Sc. and Ph.D. degrees in electronic engineering from the University of Granada, Granada, Spain, in 2002 and 2008, respectively. From 2003 to 2005, she was a Research Fellow with the Department of Electronics and Computer Technology, University of Granada, where she is currently a Tenured Professor. During a Research Fellowship, she carried out part of her research with the Department of Electrical and Computer Engineering, Florida State University, Tallahassee, FL, USA. Her work has led to the publication of 33 papers in indexed journals, more than 50 contributions to international conferences, the contribution to 11 projects in the several national and regional programs, 11 technology-transfer contracts and two patents in Spain. Her current research interests include VLSI and FPL signal processing systems, smart instrumentation for biosignal processing, new sensors and methods for sensing and the design of cryptoprocessors based on elliptic curves cryptography for its application to IoT devices.

Antonio García (Senior Member, IEEE) received a M.A.Sc. degree in electronic engineering, a M.Sc. degree in physics (majoring in electronics), and a Ph.D. degree in electronic engineering from the University of Granada, Granada, Spain, in 1995, 1997, and 1999, respectively. He received the National Award to the Best Academic Record for his M.A.Sc. degree. He was an Associate Professor with the Department of Computer Engineering of the Universidad Autónoma de Madrid, Madrid, Spain, before joining the Department of Electronics and Computer Technology of the University of Granada, where he currently serves as a Full Professor. He was also a Visiting Professor with the Department of Electrical and Computer Engineering at the FAMU-FSU College of Engineering of the Florida State University, Tallahassee, FL. He has authored more than 130 technical papers in international journals and conferences. Dr. García is a CAS and SP Society Member and he serves as reviewer and guest editor for several journals.