# A Fixed-Point Squaring Algorithm Using an Implicit Arbitrary Radix Number System

Saurabh D. Gupta
Southern Methodist University
Dallas, Texas USA

Mitchell A. Thornton
Southern Methodist University
Dallas, Texas USA

***Abstract* – A fixed-point squaring algorithm is formulated and implemented based on an approach that allows any number of bits to be computed in each iterative step. The primary contribution this new approach offers is the ability for a designer to change the area×latency product through the choice of a different radix or number of bits per subword to be processed in each iterative step. When the number of subword bits is increased, latency is reduced since fewer iterations are required whereas the area is increased due to the larger subcircuit for squaring each subword. Alternatively, choosing a smaller subword or radix decreases area at the expense of increasing latency since more iterations are required. The subword size can range from a single bit yielding a bit-serial squarer requiring *N* iterations for an *N*-bit operand or 'squarand,' to using the entire squarand resulting in a fully parallel squarer requiring no iterations. Because each *m*-bit subword can be considered a single digit in a number system with radix $2^m$, the squarer presented here can be considered a multiple-valued logic (MVL) digit-serial architecture. This methodology allows for technologies based on any radix of two or greater to be used, including emerging technologies, thus yielding a true multiple-valued logic squaring circuit. The algorithm is derived through the generalization of a Vedic technique where any arbitrary integer-valued radix is used. Prototype hardware implementations using both a standard cell ASIC and FPGA technologies are developed. The prototype circuits are analyzed in terms of required resources and throughput characteristics and compared to a well-known prior art squaring circuit.**

***Keywords – arithmetic circuit, squaring, fixed-point, digit-serial, higher radix***

## I. Introduction

Squaring is an essential arithmetic operation in many digital systems. Specialized squaring circuits have been proposed for digital signal processing applications such as image compression, pattern recognition and others [1]. Squaring is also a common atomic computation in cryptography algorithms. The increasing demand for cryptography hardware support in low power, high-speed mobile devices [2] provides motivation to devise improved hardware squaring circuit designs. Squaring circuit architecture is also commonly incorporated in graphics processors. Several general-purpose multiplier circuit designs have also been proposed based on squaring of input operands [3,4].

Designers generally have a choice between low-speed and low-area bit-serial or high-speed and high-area parallel squaring circuits. There are a few instances of 2-bit serial or quaternary Booth recoded designs available; however, to our knowledge our approach is the first generalized method that allows for the serialization of a squarand substring of any arbitrary size. When it is the case that bit-serial architectures do not meet speed requirements, but parallel versions exceed the speed requirements, a penalty in area results since the parallel architecture must be used to meet system timing requirements. A unique feature of the approach described here is that a hybrid serial/parallel squaring circuit can be formulated by partitioning the squarand into *n* substrings of *m* bits where *m* can be of any size as long as the product *nm* is equivalent to the overall squarand wordsize. Each of the *n* substrings is processed serially while the individual *m* bits comprising a substring are processed in parallel. Allowing the designer to choose the value *m* results in a squaring circuit that meets the timing specification while minimizing the overall area. In the case where $n$=1, a fully parallel squaring circuit results while the choice of $m$=1 results in a bit-serial architecture. Allowing *m* to vary from a single bit to the entire squarand wordsize provides the area/delay tradeoff that is the benefit of our squaring algorithm. Any prior art parallel squaring circuit may be used as a subcircuit in the approach described here allowing for the advantages of those past approaches to be present and enhanced by forming the overall circuit as a hybrid combination of a serial and parallel approach.

To further illustrate the contribution of our approach, consider an example where a designer wishes to implement a squaring circuit for a 64-bit squarand. A state-of-the-art 64-bit parallel squaring circuit may meet the performance requirement, or even be faster than the required speed, but it may also exceed the area requirement. In contrast, a state-of-the-art bit-serial squaring circuit may meet the area requirement, or even be much smaller than the maximum allowable area, but it may not meet the performance requirement. It would be desirable in either instance to tradeoff the area versus the performance. In the past, a designer with this difficulty would be forced to find a new squaring circuit architecture that met both requirements. Our new approach offers a methodology whereby the designer can tradeoff the competing area versus delay characteristics through the choice of an appropriate radix value or substring size *m*. In essence, the squaring circuit architecture described here allows for a designer to optimize the area×latency product of the squaring circuit. For example, the designer may be able to meet the performance requirement by choosing $m$=16 since a 16-bit squaring subcircuit is relatively fast as compared to the bit-serial approach, but it is not as fast as the 64-bit parallel squaring circuit. From the point of view of MVL, choosing $m$=16 is tantamount to using digits from a radix-$(65{,}536)_{10}$ number system since each 16-bit substring can be considered a single radix-$(65{,}536)_{10}$ digit. This hypothetical solution also provides a substantial reduction in

area since the combination of a single 16-bit parallel squaring circuit with the digit-serial "wrapper" circuitry described in this paper is much smaller than a 64-bit parallel squaring circuit. The digit-serial "wrapper" circuitry is specific to the results described here but the parallel squaring subcircuit may be of any desired architecture.

The ability for a designer to meet specific performance requirements is crucial for some applications of squaring circuits. In particular, for those systems with real-time constraints, it is essential that system component delays do not cause deadlines to be missed. One example of such a system is in modern communication systems that utilize squaring-intensive subsystems such as Viterbi decoders [21,22]. In such systems, it is mandatory to meet the delay requirements and a highly desirable requirement to minimize area. The approach described in this paper allows for a squaring circuit to be formulated such that these requirements are met without excessive slack. Another application that has a strict performance requirement is that of adaptive sample rate digital notch filters [23]. These notch filters can be implemented with squaring operations and the delay of the squaring unit must not cause the system to violate the system sample rate specifications, while at the same time it is desirable for the squaring units to utilize as few transistors as possible. Yet another application is that of direct digital frequency synthesis circuits that have numerous applications including digital radio implementations. In these applications, the area-latency product is an important optimization parameter and to meet these stringent requirements, methods that provide approximate squaring values are often used such as the CORDIC algorithm for rectangular-to-polar conversions [24]. Rectangular-to-polar conversions are very squaring intensive operations and the squaring method provided in this paper is directly applicable for optimization of the area×latency product allowing approximate methods such as CORDIC to be replaced with our exact method.

The other significant advantage of our approach is that true Multiple-Valued Logic (MVL) squaring circuits can be designed using emerging MVL circuit components. Circuits utilizing components with non-binary (or powers of 2) radices can be designed with this algorithm. Because we did not have access to FPGAs nor standard cell libraries that are non-binary, our experimental results are limited to using binary circuitry with digits formed as a grouping of $m$ bits. However, should FPGAs or standard cell libraries with elements based on a radix of $p$, where $p$ is not a power of two become available, the technique described here is applicable.

Bit-serial designs have been proposed in [4,5,6,7] which can be considered a case where $m$=1 and $n$ is the squarand wordsize in units of bits. All these designs require $2nm$ iterations to generate a squared result of $2nm$-bits in size. These designs have seen improvements in area and delay through various improved architectures, but they still have limitations on throughput due to their bit-serial nature.

Bit-parallel designs have also been proposed to obtain approximate squaring results [8-12][18-23]. Most parallel squaring designs make use of the fact that generation of the result, $\alpha^2$, can be accomplished through the formulation of one-half of the number of partial products as compared to the use of a general multiplication circuit [1].

Several designs have been published based on Booth recoding and Booth folding techniques that use adder trees for accumulation of the partial products of $\alpha^2=\alpha\times\alpha$ [9,10,11,14]. Higher-radix parallel-array designs with a left-to-right dual recoding method is described in [12] and with a right-to-left significant bit recoding of the input operand described in [13].

Bit-parallel architectures may make use of some form of Booth recoding and efficient multi-operand tree addition structures. Though these approaches do not suffer from the performance limitations inherent in bit-serial architectures and can output more than one or two resultant bits per cycle, a relatively large amount of circuitry is required to provide support for the multi-operand addition circuit used to accumulate the partial products. Some notable recent parallel squaring circuits include [18] and [19]. [18] provides a method that produces the square as an approximation that can be iteratively corrected to any degree of required accuracy thus allowing for a speed versus accuracy tradeoff. [19] allows for a squaring circuit based on a radix-10 Vedic method whose performance is enhanced through retiming. Both of these are binary methods whereas our approach can utilize any arbitrary radix. These, or other approaches, can be used as sub-circuit components in our approach for computing the square of an individual digit if desired by the designer.

Arithmetically, the technique presented here assumes the squarand is represented as a higher-radix digit string consisting of $n$ radix-$\beta$ digits. Each radix-$\beta$ digit is a member of the canonical set $\{0,1,\ldots,\beta\text{-}1\}$. Each iteration of the squaring algorithm produces two radix-$\beta$ digits in the resulting output squared value. The squaring operation is completed after $n$ iterations have occurred, thus this technique is properly described as an MVL digit-serial technique. The final squared output value is in the form of a $2n$ string of radix-$\beta$ digits.

When the target technology is conventional digital electronics, the allowable choices of radices are restricted to $\beta=2^m$. This restriction is only due to the fact that we are using digital circuitry since the theory behind the approach is general and supports any arbitrary radix that is a positive integer of value two or greater. When $\beta=2^m$, the squaring circuit produces a $2nm$-bit length result, $\alpha^2$, based on a corresponding input operand (squarand) $\alpha$ of $nm$-bits in length. The circuit produces $2m$ bits in the output $\alpha^2$ during each iterative step. By considering an $m$-bit grouping within the squarand $\alpha$ as representing a single radix-$2^m$ digit, the circuit can be considered a MVL digit-serial implementation that produces two MVL digits per iteration in the form of a single $2m$-bit string.

This digit-serial architecture allows for a tradeoff between bit-serial and parallel architectures by allowing for each operand digit to be implicitly represented by $m$ bits. Because $2m$ bits of the result are computed in each iterative step, varying $m$ can yield more or less parallelism while inversely affecting the required circuit area. Thus, a minimal area circuit can be realized when $m$ is small (bit-serial for the case $m$=1) and a large parallel circuit results at the other extreme when $m$ is set to the wordsize of the squarand. It is envisioned that designers will choose an appropriate value of $m$ such that performance requirements are met while minimizing the amount of circuitry required.

The paper is organized as follows. Section II provides the theoretical results that are used as the basis of the algorithm. Section III contains the foundations of the algorithm and Section IV contains a statement of the algorithm and an explanation of how the theoretical results from the previous section relate to the method. In Section V, we describe how the algorithm is applied to squarands expressed as fixed-point bit strings and how the algorithm is implemented as a synchronous hardware logic circuit. Section VI contains an evaluation of the prototype implementation when synthesized as both a programmable logic circuit and as a standard cell ASIC. The paper concludes in Section VII where a summary of the approach and directions for future related work is given.

## II. Background of Approach

The following notation is used in the description of the digit-serial fixed-point squaring algorithm.

- $\beta$ represents the radix or base of a number system. We restrict $\beta$ to the set of integers such that $\beta>1$. For compatibility with conventional binary digital logic, $\beta=2^m$ and $m\geq 1$; however, this restriction is not theoretically required and the technique is applicable to multiple-valued logic circuits.
- The 'radix polynomial' form of a value $\alpha$ is written as an $n$-term polynomial of the form [14]: $\alpha = a_{n-1}\beta^{n-1}+ a_{n-2}\beta^{n-2}+\ldots+ a_2\beta^2+ a_1\beta^1+ a_0\beta^0$
- A value $\alpha$ can also be represented in the radix-$\beta$ number system in the form of a positional string of $n$ characters denoted by $\alpha=[a_{n-1}a_{n-2}\ldots a_2a_1a_0]$. For clarity, the character strings denoting the positional digit representations of a value $\alpha$ are enclosed by square brackets with leading zeros omitted. The digits $a_i$ are the coefficients of the radix-polynomial form and their position within the string inherently denotes the exponent of the radix $\beta$.
- Each character $a_i$ in a positional string representing a value is referred to as a 'digit' regardless of the radix of the number system. Binary digits may alternatively be referred to as 'bits.'
- Digits are restricted to the integral numbers and are members of the set $\{a_i\in\mathbb{Z}\}$.
- Where necessary for clarity, digit strings are subscripted by the radix $\beta$ of the particular number system being used, $\alpha=[a_{n-1}a_{n-2}\ldots a_2a_1a_0]_\beta$.
- LSD($\alpha$,$k$) and MSD($\alpha$,$k$) are operators that yield $k$ least significant or most significant digits in a digit string representing the value $\alpha$. LSD($\alpha$,1) represents the least significant digit of $\alpha$, LSD($\alpha$,1)=$a_0$. Likewise the most significant digit is given as MSD($\alpha$,1)=$a_{n-1}$.
- {A,B,C} denotes concatenation of the content of registers A, B, and C which can be of any size and whose individual sizes may differ.
- SHL(A,$k$,B) denotes the operation of shifting the content of register A to the left by $k$ bits and setting the least significant $k$ bits to the content of register B. A can be of any size greater than or equal to the size of B and B must be of size $k$.
- SHR(A,$k$,B) denotes the operation of shifting the content of register A to the right by $k$ bits and setting the most significant $k$ bits of A to the content of register B. A can be of any size greater than or equal to the size of B and B must be of size $k$.
- A←B denotes the operation of setting the content of register A with that of register B. A and B must be of the same size and the original content of A is destroyed.

The initial motivation for the theory of the method developed here arises from the Vedic technique whose Sanskrit name is '*Ekādhikena Pūrvena.*' Loosely translated '*Ekādhikena Pūrvena*' is "[by] one more than the previous one." This technique describes how the square of a decimal integer $\alpha$ may be easily obtained, when $\alpha$ is of the special form where LSD($\alpha$,1)=5. Using the notation previously defined, the square of a two-digit radix-10 value $\alpha$ with LSD($\alpha$,1)=5 can be formed as $\alpha^2=\{[\text{MSD}(\alpha,1)\times(\text{MSD}(\alpha,1)+1)],[(\text{LSD}(\alpha,1)^2]\}$. To illustrate '*Ekādhikena Pūrvena*', consider the following example.

***Example* 1**: Determine the square of decimal squarand $45_{10}$ using the technique of '*Ekādhikena Pūrvena.*' It is noted that LSD(45,1)=5 and $5^2=25$. Thus, the two least significant digits of $45^2$ are the string $[25]_{10}$. Since MSD(45,1)=4, the two most significant digits are formed by multiplying MSD(45,1)+1=5 with MSD(45,1)=4 yielding the string $4\times5=[20]_{10}$. $45^2$ is then obtained by the concatenation $\{[20]_{10},[25]_{10}\}=2025_{10}$. □

While the method is interesting, it is limited to cases where the squarand is a radix-10 value with the least significant digit happening to be exactly one-half of the radix value, $\beta/2=10/2=5$. Lemma 1 generalizes '*Ekādhikena Pūrvena*' to account for the case where the squarand is represented in an arbitrary radix-$\beta$ number system. For convenience in the derivation of the result of Lemma 1 we first define the radix-$\beta$ value $A$.

***Definition* 1**: The radix-$\beta$ value $A$ is defined as $A=\alpha-a_0$. Expressed as a radix-$\beta$ positional $n$-digit string, $A=[a_{n-1}a_{n-2}\ldots a_2a_1 0]_\beta$. Thus, $A$ can be easily formed by replacing LSD($\alpha$,1)=$a_0$ with the zero digit $[0]_\beta$.

□

***Lemma* 1:** Consider a value expressed as a digit string $\alpha=[a_{n-1}a_{n-2}\ldots a_2a_1a_0]_\beta$ where $a_0=(\beta/2)$. The square $\alpha^2$ may be expressed as shown in Equation (1).

$$\alpha^2=\left(\frac{A}{\beta}\right)^2\beta^2+\left(\frac{A}{\beta}\right)\beta^2+\left(\frac{\beta}{2}\right)^2 \tag{1}$$

***Proof*:** Expressing $\alpha$ in radix polynomial form:

$$\alpha = a_{n-1}\beta^{n-1}+a_{n-2}\beta^{n-2}+\ldots+a_2\beta^2+a_1\beta^1+\frac{\beta}{2} \tag{2}$$

Since LSD($\alpha$,1)=$a_0=\beta/2$, we express $\alpha=A+\beta/2$ and $\alpha^2$ becomes:

$$\begin{aligned}\alpha^2&=\left(A+\frac{\beta}{2}\right)^2=A^2+A\beta+\left(\frac{\beta}{2}\right)^2\\&=\left(\frac{A}{\beta}\right)^2\beta^2+\left(\frac{A}{\beta}\right)\beta^2+\left(\frac{\beta}{2}\right)^2\end{aligned} \tag{3}$$

□

The result of Lemma 1 can be used as the basis of a squaring algorithm where the second two terms of the right-

hand side of Equation (1) are calculated in each iterative step, the first term is used for subsequent iterations, and the results are accumulated at each step. The algorithm clearly converges since subsequent iterative steps use an operand with one less digit in the operand digit string representation. The $\beta^2$ factor in the first term is accounted for by implementing a shifting operation during the accumulation step.

Unfortunately, Equation (1) only holds for the special case where the squarand and subsequent intermediate arguments happen to exhibit the property LSD($\alpha$,1)=$a_0$=$\beta$/2. Lemma 2 generalizes Equation (1) for the case where LSD($\alpha$,1)≠$\beta$/2 in the original squarand and for subsequent iterative operands.

For convenience in the derivation of the result of Lemma 2, we define a signed single recoded digit value $r$ in the radix-$\beta$ number system referred to as the 'residual.'

***Definition* 2**: The residual value $r$ is the difference between $a_0$ and $\beta$/2 given by $r=a_0-\beta/2$. In terms of a digit string representation, $r$ is in the form of a recoded signed radix-$\beta$ value in general. Restricting the radix $\beta$ to even values of two or greater causes $r$ to be of the form of a single signed radix-$\beta$ digit and may be desirable for the purposes of implementation; however this restriction is not theoretically required.

$$r \in \left\{\left(0-\frac{\beta}{2}\right),\left(1-\frac{\beta}{2}\right),...,(\beta-1)-\frac{\beta}{2}\right\} \quad \square$$

***Lemma* 2:** Consider a value expressed as a digit string $\alpha=[a_{n-1}a_{n-2}\ldots a_2a_1a_0]_\beta$. $\alpha^2$ may be expressed as:

$$\alpha^2=\left(\frac{A}{\beta}\right)^2\beta^2+\left(\frac{A}{\beta}\right)\beta^2+\left(\frac{\beta}{2}\right)^2+2\left(A+\frac{\beta}{2}\right)r+r^2 \tag{4}$$

***Proof*:** Expressing $\alpha$ in radix polynomial form

$$\alpha=a_{n-1}\beta^{n-1}+a_{n-2}\beta^{n-2}+...+a_2\beta^2+a_1\beta^1+\left(\frac{\beta}{2}+r\right) \tag{5}$$

Since $a_0=\beta/2+r$, we can express $\alpha=A+(\beta/2+r)$ and $\alpha^2$ becomes

$$\begin{aligned}\alpha^2&=\left[A+\left(\frac{\beta}{2}+r\right)\right]^2=A^2+2A\left(\frac{\beta}{2}+r\right)+\left(\frac{\beta}{2}+r\right)^2\\&=A^2+A\beta+2Ar+\left(\frac{\beta}{2}\right)^2+r\beta+r^2\\&=\left[A^2+A\beta+\left(\frac{\beta}{2}\right)^2\right]+(2A+\beta)r+r^2\\&=\left(\frac{A}{\beta}\right)^2\beta^2+\left(\frac{A}{\beta}\right)\beta^2+\left(\frac{\beta}{2}\right)^2+2\left(A+\frac{\beta}{2}\right)r+r^2\end{aligned} \tag{6}$$

$\square$

Equation (4) expresses the square $\alpha^2$ when $\alpha$ is represented as any arbitrary digit string in the radix-$\beta$ number system. It is noted that the form of Equation (4) is the same as that of the special case where $a_0=\beta/2$ as given in Equation (1) and augmented with a correction term $\varepsilon$ defined as:

$$\varepsilon=2\left(A+\frac{\beta}{2}\right)r+r^2 \tag{7}$$

## III. Basis of Algorithm

Equation (4) is used to formulate our digit-serial squaring algorithm. The motivation for forming this algorithm is that the choice of radix $\beta$ allows for a trade-off in logic circuit area versus throughput performance in the computation of $\alpha^2$ when $\alpha$ is represented as a binary bit string. Higher values of $\beta$ allow more bits to be produced per iterative step in the resulting representation of $\alpha^2$. A tradeoff occurs in that the amount of computation or logic required at each iterative step increases for higher radix values. This tradeoff is very useful for arithmetic circuit designers since it allows them to formulate a squaring circuit architecture that meets timing requirements without greatly exceeding them and therefore incurring an area penalty.

In the basis of the algorithm as stated here, we assume that the squarand is of the form of a binary bit string. Intermediate computations can be efficiently implemented when we restrict the radix $\beta$ to be of the form $\beta=2^m$ where $m$ is a positive integer $m\geq2$. Efficiency results since $\beta=2^m$ allows each higher radix digit in the string representing $\alpha$ to be equivalent to an $m$-bit substring within $\alpha$. $\alpha$, in terms of a higher-radix digit string, is simply the concatenation of the disjoint $m$-bit substrings of $\alpha$ in binary form where LSD($\alpha_\beta$,1) is the least significant $m$ bits, the subsequent next significant higher-radix digit is represented by the next group of $m$ bits to the left of LSD($\alpha$,1), and so on.

We point out that the results of Lemma 1 and 2 hold for any general radix value and the restriction that $\beta=2^m$ is only used for convenience in formulating the squaring algorithm when squarands are given as a binary bit string. This case is of particular interest in our implementation since we are targeting computer arithmetic circuits and algorithms implemented with binary switching logic. However, future technologies may employ non-binary, multiple-valued switching elements [15] and this technique is equally applicable for such non-binary technologies.

For compatibility with conventional binary digital circuitry, we rewrite Equation (4) with the restriction that $\beta=2^m$ and define some of the individual terms on the right-hand side of the equation to be denoted with $T_1$, $T_2$, and $T_3$.

$$\begin{aligned}\alpha^2&=\left(\frac{A}{\beta}\right)^2 2^{2m}+\left[\left(\frac{A}{\beta}\right)2^{2m}+\left(\frac{\beta}{2}\right)^2\right]+\left[2\left(A+\frac{\beta}{2}\right)r\right]+\left[r^2\right]\\&=\left(\frac{A}{\beta}\right)^2 2^{2m}+T_1+T_2+T_3\end{aligned} \tag{8}$$

The terms $T_1$, $T_2$ and $T_3$ are explicitly defined in the following equations.

$$T_1=\left(\frac{A}{\beta}\right)2^{2m}+\left(\frac{\beta}{2}\right)^2$$

$$T_2=2\left(A+\frac{\beta}{2}\right)r$$

$$T_3=r^2$$

The algorithm is implemented by iteratively computing terms $T_1$, $T_2$ and $T_3$ and accumulating them with previously computed values. The particular partitions, $T_1$, $T_2$ and $T_3$ were chosen to allow for a pipelined hardware implementation of the algorithm. Subsequent iterations use $A/\beta$ from the $(A/\beta)^2$ term in Equation (8) as next squarand. The operand for each iterative step, $A/\beta$, is a digit string containing one less digit than the squarand in the previous step, thus the algorithm requires O($n$) iterations to complete. The $2^{2m}$ shifting factor of the first term in Equation (8) indicates that two radix-$\beta$ digits (two $2m$-length bit strings) in $\alpha^2$ are produced after each iteration is completed. The resulting digits in $\alpha^2$ are produced in the order of the lesser significant digits first (right-to-left fashion).

Before stating the algorithm in Register Transfer Level (RTL) form, several observations are noted and used to more efficiently implement the computation of the three terms $T_1$, $T_2$ and $T_3$.

***Observation* 1**: The term $A/\beta$ is efficiently obtained by shifting the digit string representing $\alpha$ one position to the right and discarding $a_0$, $A/\beta=[a_{n-1}a_{n-2}\ldots a_2a_1]_\beta$. □

***Observation* 2**: Values that are multiplied by a factor of $\beta=2^{km}$ may be easily obtained by shifting the value to left by $km$ bit positions and inserting a radix-$\beta$ zero digit place holder $[0]_\beta$ for the vacated least significant digits. □

***Observation 3***: The term $\beta/2$ is always of the form of a single radix-$\beta$ digit. Expressed as an $m$-bit binary string $\beta/2=[10\ldots0]_2$. □

***Observation 4***: The term $(\beta/2)^2$ is always of the form of two radix-$\beta$ digits, $[q_1q_0]_\beta$, with the most significant digit of value $q_1=\text{MSD}((\beta/2)^2,1)=\beta/4=[010..0]_2$ and the least significant digit of value $q_0=\text{LSD}((\beta/2)^2,1)=0=[0\ldots0]_2$. Hence, expressed as a $2m$-bit binary string, $(\beta/2)^2=\{[010\ldots0]_2, [000\ldots0]_2\}$. □

Term $T_1$ is computed in a single operation. Making use of Observations 1 and 2, the value $(A/\beta)2^{2m}$ is obtained by forming the digit string $[a_{n-1}a_{n-2}\ldots a_2a_100]_\beta$. Furthermore by Observation 4, $\beta/2^2$ can always be expressed as two radix-$2^m$ digits ($2m$ bits) denoted as $[q_1q_0]_\beta$. Thus, $T_1$ is obtained by forming the string $[a_{n-1}a_{n-2}\ldots a_2a_1q_1q_0]_\beta$. From Observation 4, $q_1=\beta/4$ and $q_0=0$ so that $(\beta/2)^2=[q_1q_0]_\beta=[(\beta/4)0]_\beta$. Thus, the digit string representation for $T_1$ is $[a_{n-1}a_{n-2}\ldots a_2a_1(\beta/4)(0)]_\beta$.

Term $T_2$ is computed by first forming a digit string representing $2(A+\beta/2)$ and then multiplying this string with the single radix-$\beta$ digit $r$. Using Observations 1, 2, and 3, $A=[a_{n-1}a_{n-2}\ldots a_2a_10]_\beta$ and $\beta/2$ is always represented as a single unsigned radix-$2^m$ digit ($m$-bit string), $\beta/2=[10\ldots0]_2$. Therefore, $(A+\beta/2)=[a_{n-1}a_{n-2}\ldots a_2a_1(\beta/2)]_\beta$. To account for the multiplicative factor of 2, the $(A+\beta/2)=[a_{n-1}a_{n-2}\ldots a_2a_1(\beta/2)]_\beta$ digit string is then shifted by one bit position to the left resulting in $2(A+\beta/2)$. We note that the multiplicative factor 2 would in general need to be implemented through the use of an addition operation, $2(A+\beta/2)=(A+\beta/2)+(A+\beta/2)$, when a higher-valued radix $\beta$ is used that is not an integral power of two since this can be considered a 'fractional digit shift.'

The final step in the formation of term $T_2$ involves the multiplication of $2(A+\beta/2)=[a_{n-1}a_{n-2}\ldots a_2a_1(\beta/2)]_\beta$ by the single radix-$2^m$ signed digit, $r=a_0-\beta/2$. Because $r$ is a single signed digit value, this multiplication can be accomplished with a minimal amount of computation or circuitry as compared to a general-purpose multiply operation or circuit. Clearly, as the value $m$ is increased resulting in a higher valued radix, $2^m$, both computational complexity and overall algorithm throughput increase. The actual implementation of the multiplication by $r$ is dependent upon the value $m$ and based upon the desired throughput and area constraints for a given realization of the algorithm. Relatively small values of $m$ generally allow for a simple logic circuit or lookup table to be used whereas larger values of $r$ utilize any of a variety of $m$-bit parallel or accumulation tree multiplier subcircuits.

Term $T_3=r^2$ requires the computation of the square of the residual value $r$. The implementation of this computation is also dependent upon the size of $m$, which dictates the number of bits required to represent a radix-$2^m$ digit. For smaller values of $m$, the direct calculation of $r^2$ can be very efficiently implemented as a small combinational logic circuit or through a lookup table. As $m$ increases, the computation of $r^2$ becomes more complex and any other methods such as the bit-parallel methods cited previously may be employed. We note that even for large values of $m$, the computation of $T_3=r^2$ can be accomplished in parallel with the computation of the other two terms $T_1$ and $T_2$ since the accumulation of $T_1+T_2+T_3$ with the overall result can occur at the end of each iterative step. For this reason, it is not necessary to choose the absolute highest speed parallel squaring circuit to compute $r^2$ and corresponding area penalties are not overly severe.

After terms $T_1$, $T_2$ and $T_3$ are formulated, they are summed together and accumulated with the previous result. The accumulation takes into account the process of multiplying subsequent iterative operands by $2^{2m}$ and the fact that two independent radix-$\beta$ digits (or, $2m$ bits) of the final result are produced at each iterative step. This can be implemented in a variety of ways. We choose to initialize a final result register to zero with size $2nm$ bits where $n$ is the number of radix-$\beta$ digits representing $\alpha$ and $m$ implicitly denotes the radix. The final operation of each iterative step of the algorithm is to shift the result register $2m$ bits to the right and insert the $2m$ least significant bits of $T_1+T_2+T_3$ into the most significant positions of the shifted result register. Insertion of the two radix-$2^m$ digits in the most significant portion of the result register instead of performing a multi-bit left shift before adding them to the previously accumulated result allows the algorithm to be implemented without the need for a inclusion of a multi-bit left shift operation or the use of a barrel shifting circuit in a hardware realization. This is an important aspect of the accumulation process since a multi-bit left-shift operation is considerably more complex as compared to a fixed length ($2m$ bit) right-shift operation.

The algorithm uses an iteration index $i$ to determine if all digits of the squarand have been produced. For an $n$-digit radix-$\beta$ squarand, the squared result consists of $2n$ digits. Since two digits are produced per iterative step, the index $i$ ranges from zero to $(n/2)$-1. Initially, when $i=0$, $\alpha$ serves as the squarand. During intermediate computations, when $0<i<n/2$, the algorithm iterates and sets the intermediate squarand to $A/\beta$. In the final iterative step, the squarand argument becomes 0; however, this step is required since the residual $r$ may not be zero-valued and must be included in the resultant $\alpha^2$ value.

Conceptually, the residual $r$ is a recoded signed value and $m+1$ bits are needed to account for the sign if $r$ is expressed explicitly. Depending upon the definition of the residual, $r$ can take on integer values in either of the ranges $[-(\beta/2),(\beta/2-1)]$ (as is the case in this formulation) or $[-(\beta/2)+1,(\beta/2)]$. However, since there is a one-to-one relationship between $a_0$ and $r$ (since $r=a_0-\beta/2$), we use the $m$-bit string representing $a_0$ to represent the corresponding $r$ value thus allowing $r$ to be encoded as an $m$-bit string. This is in essence an encoding of the recoded residual, $r$.

## IV. Algorithm Implementation

We use the results of the previous sections to specify the method as an RTL algorithm that is suitable for

implementation in hardware, software, or a combination of the two target technologies. The RTL implementation makes use of several registers. For succinctness, we define the registers used within the algorithm statement in Table 1.

**Table 1: Registers Used in Squaring Algorithm**

| name | size (bits) | content |
|---|---|---|
| AB | ($n$-1)m | $A/\beta$ |
| RES | 2$nm$ | $\alpha^2$ |
| i | $\log_2(n)$ | iteration index |
| R | $m$ | residual $r$ encoded as LSD($\alpha$,1) |
| ACC | 2$nm$ | $T_1+T_2+T_3$ |
| T1 | 2$nm$ | $T_1$ |
| T2 | 2$nm$ | $T_2$ |
| T3 | 2$nm$ | $T_3$ |
| B2 | $m$ | $\beta/2$ |
| B4 | 2$m$ | $(\beta/2)^2=(\beta^2/4)=[(\beta/4)0]_\beta$ |

A statement of the algorithm is provided in Figure 1 using the RTL operations previously defined. Intermediate locations within the algorithm are denoted by labels in the form 'STEP $k$.' The labels are included for convenience in referring to certain portions of the algorithm and they also indicate clock boundaries in a hardware implementation since the results of STEP $k$-1 are registered before computation can occur in STEP $k$. As an example, the T2←{AB,B2} operation of STEP 3 must complete before the T2←SHL(T2,1,$[0]_2$) operation of STEP 4 can proceed. Partitioning the computation of term $T_2$ into multiple intermediate registered operations is an instance of circuit pipelining and allows for the overall circuit clock speed to be increased in a hardware realization of the algorithm.

**INPUT**:
  $\alpha$: $nm$-bit fixed-point squarand
  $m$: $\log_2(\beta)$-bit value, indicates working radix $2^m$
**OUTPUT**:
  $\alpha^2$: 2$nm$-bit value in register RES
STEP 1:
  i←0 /* iteration index */
  RES←$[0\ldots0]_2$ /* initialize result register */
  B2←$[10\ldots0]_2$ /* $m$-bits with MSb=1 */
  B4←$[010\ldots0]_2$ /* 2$m$-bits with MSbs=01 */
  AB←$\alpha$ /* squarand value */
STEP 2:
  R←LSD(AB,$m$) /* encode $r$ as LSD(AB,1) */
  AB←SHR(AB,$m$,$[0..0]_2$) /* MS squarand digits */
STEP 3:
  T1←{AB,B4,$[0..0]_2$} /* form $T_1$, $m$ LSbs=0 */
  T2←{AB,B2} /* form $A+\beta/2$ */
  T3←$r\times r$ /* compute square, uses $a_0$ in R */
STEP 4:
  T2←SHL(T2,1,$[0]_2$) /* form $2(A+\beta/2)$ */
  ACC←T1+T3 /* form $T_1+T_3$ */
STEP 5:
  T2←T2×$r$ /* form $2(A+\beta/2)b$, uses $a_0$ in R */
STEP 6:
  ACC←ACC+T2 /* accumulate $T_1+T_2+T_3$ */
STEP 7:
  RES←SHR(RES,2$m$,LSD(ACC,2)) /* update result */
  i←i+1 /* increment iteration counter */
STEP 8:
  if (i=$n$) /* check iteration count */
    HALT /* computation complete */
  else
    GO TO STEP 2 /* further iteration required */

**Figure 1: Iterative Squaring Algorithm**

In terms of required computational resources, the algorithm requires circuitry or memory to perform shifting, bit-string concatenation, ($nm$+$m$)-bit and ($nm$+$m$+1)-bit operand addition, $m$×2$nm$-bit multiplication, and $m$-bit operand squaring. While ($nm$+$m$)-bit and ($nm$+$m$+1)-bit operand addition operations are required in STEPs 4 and 6, it is noted that a single ($nm$+$m$+1)-bit addition circuit can be used since these sums may be formed sequentially allowing for reuse of a single ($nm$+$m$+1)-bit adder. The multiplication and single-digit squaring operations can be implemented in a variety of forms although it is noted that for relatively small sizes of the digit operands ($m$ bits) very compact and fast circuits such as lookup tables are a practical choice. With respect to throughput, the algorithm requires $n$ iterations producing 2$m$ bits of $\alpha^2$ during each iterative step. Therefore, the algorithm has temporal complexity equivalent to O($n$) as expected since it is a digit-serial method over a squarand composed of $n$ digits. The performance increase occurs for higher radix values $\beta$>2 allowing each of the sequentially produced $n$ digits to comprise $m$ bits.

## V. Quaternary Radix Squaring Circuit

To demonstrate and evaluate the digit-serial squaring algorithm, we designed and implemented a synchronous digital logic circuit using a quaternary radix, $\beta=2^2=4$. This choice of radix allows for comparison to other squaring circuits based on radix-4 Booth recoding and provides an intermediate solution between bit-serial and bit-parallel realizations. The circuit architecture is of the form of a clocked synchronous controller with a corresponding datapath subcircuit that implements the operations specified in the algorithm. A block diagram of the datapath is shown in Figure 2.

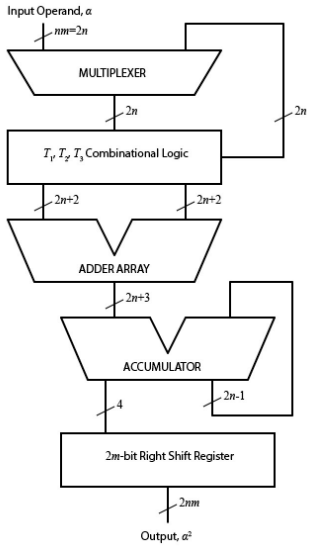


**Figure 2: Quaternary Squaring Circuit Datapath**

The datapath element labeled "Combinational Logic" is implemented based on simplifications in the formation of the intermediate terms $T_1$, $T_2$, and $T_3$ and their various sums. These simplifications exploit the choice of using $\beta$=4 as an implicit operand radix and allow for the computation of the intermediate terms $T_1$, $T_2$, and $T_3$ to be implemented with a reduced and simplified set of RTL operations. The terms $T_1$, $T_2$, and $T_3$ are chosen based on implementation decisions and

other intermediate terms are possible and should be considered based on desired pipeline depth and the target technology characteristics.

As an aid in explaining the quaternary radix specific optimizations, the notation in Definition 3 is used to represent bit strings.

***Definition* 3**: A single quaternary digit $[a_k]_4$ can, in general, be written as a two-bit binary string $[b_{2k+1}b_{2k}]_2$ where $\{b_i \in \mathbb{B}\}$ and $\mathbb{B}=\{0,1\}$. □

Using Definition 3, we evaluate the various intermediate terms and their sums for different cases of the least significant digit of the squarand, $a_0 \in \{0,1,2,3\}$. Term $T_1$ is independent of the value of $a_0$ and is always a bit string of length $2n+2$ expressed as:

$$T_1=[a_{n-1}a_{n-2}\ldots a_2a_110]_4=[b_{2n-1}b_{2n-2}b_{2n-3}b_{2n-4}\ldots b_5b_4b_3b_20100]_2$$

***Case* 1**: $a_0=[0]_4$ resulting in the residual $r=[-2]_4$, thus $T_3=r^2=[10]_4=[0100]_2$. Term $T_2$ can be expressed as:

$$\begin{aligned} T_2 &= 2\left(A+\frac{\beta}{2}\right)r = 2r\times[a_{n-1}a_{n-2}..a_2a_12]_4 \\ &= -[10]_4\times[a_{n-1}a_{n-2}..a_2a_12]_4 \\ &= -[a_{n-1}a_{n-2}..a_2a_120]_4 \end{aligned}$$

Combining the terms:

$$\begin{aligned} T_1+T_2+T_3 &= [a_{n-1}a_{n-2}\ldots a_2a_110]_4 \\ &\quad -[a_{n-1}a_{n-2}\ldots a_2a_120]_4+[10]_4 \\ &= [0\ldots0]_4=[00\ldots00]_2 \end{aligned}$$

□

***Case* 2**: $a_0=[1]_4$ resulting in the residual $r=[-1]_4$, thus $T_3=r^2=[01]_4=[0001]_2$. Term $T_2$ can be expressed as:

$$\begin{aligned} T_2 &= 2\left(A+\frac{\beta}{2}\right)r = 2r\times[a_{n-1}a_{n-2}\ldots a_2a_12]_4 \\ &= -[2]_4\times[a_{n-1}a_{n-2}\ldots a_2a_12]_4 \\ &= -[0b_{2n-1}b_{2n-2}\ldots b_3b_2100]_2 \end{aligned}$$

Combining the terms:

$$\begin{aligned} T_1+T_2+T_3 &= [b_{2n-1}b_{2n-2}\ldots b_3b_20100]_2 \\ &\quad -[0b_{2n-1}b_{2n-2}\ldots b_3b_2100]_2+[0001]_2 \\ &= [0b_{2n-1}b_{2n-2}\ldots b_3b_2001]_2 \end{aligned}$$

□

***Case* 3**: $a_0=[2]_4$ resulting in the residual $r=[0]_4$, thus $T_3=r^2=[00]_4=[0000]_2$. Term $T_2$ can be expressed as:

$$\begin{aligned} T_2 &= 2\left(A+\frac{\beta}{2}\right)r = 0\times[a_{n-1}a_{n-2}\ldots a_2a_12]_4 \\ &= [00\ldots00]_2 \end{aligned}$$

Combining the terms:

$$\begin{aligned} T_1+T_2+T_3 &= [b_{2n-1}b_{2n-2}\ldots b_3b_20100]_2 \\ &\quad +[00\ldots00]_2+[0000]_2 \\ &= [b_{2n-1}b_{2n-2}\ldots b_3b_20100]_2 \end{aligned}$$

□

***Case* 4**: $a_0=[3]_4$ resulting in the residual $r=[1]_4$, thus $T_3=r^2=[01]_4=[0001]_2$. Term $T_2$ can be expressed as:

$$\begin{aligned} T_2 &= 2\left(A+\frac{\beta}{2}\right)r = 2r\times[a_{n-1}a_{n-2}\ldots a_2a_12]_4 \\ &= [2]_4\times[a_{n-1}a_{n-2}\ldots a_2a_12]_4 \\ &= [0b_{2n-1}b_{2n-2}\ldots b_3b_2100]_2 \end{aligned}$$

For this case, the sum $T_2+T_3$ can be formed directly and it is subsequently combined with term $T_1$ using the addition circuit. $T_2+T_3$ is formed as:

$$\begin{aligned} T_2+T_3 &= [0b_{2n-1}b_{2n-2}\ldots b_3b_2100]_2+[0001]_2 \\ &= [0b_{2n-1}b_{2n-2}\ldots b_3b_2101]_2 \end{aligned}$$

□

Table 2 contains a summary of the results of the intermediate terms and their various sums in terms of values of the least significant digit of the operand at each iterative step.

**Table 2: Radix-4 Optimizations**

| LSD($\alpha_4$,1) | Intermediate Term | Value |
|---|---|---|
| 0 | $T_1+T_2+T_3$ | $[0\ldots0]_2$ |
| 1 | $T_1+T_2+T_3$ | $[0b_{2n-1}b_{2n-2}\ldots\ b_3\ b_2001]_2$ |
| 2 | $T_1+T_2+T_3$ | $[b_{2n-1}b_{2n-2}\ldots\ b_3\ b_20100]_2$ |
| 3 | $T_1$ | $[b_{2n-1}b_{2n-2}\ldots\ b_3\ b_20100]_2$ |
| 3 | $T_2+T_3$ | $[0b_{2n-1}b_{2n-2}\ldots\ b_3\ b_2101]_2$ |

The datapath element labeled "Combinational Logic" in Figure 2 makes use of the results in Table 3 and outputs the two $2n+2$ values that are summed in the adder array resulting in $T_1+T_2+T_3$. For the cases $a_0 \in \{0,1,2\}$, $T_1+T_2+T_3$ is formed directly in the combinational logic block and is input to the adder array on the leftmost input bus with the rightmost input set to the $2n+2$ bit string $[00\ldots00]_2$. The adder array is only required for the case $a_0=3$, where the leftmost input is the bit string $[b_{2n-1}b_{2n-2}\ldots\ b_3\ b_20100]_2$ and the rightmost input is $[0b_{2n-1}b_{2n-2}\ldots\ b_3\ b_2101]_2$. Figure 3 is a logic diagram of the combinational logic block.

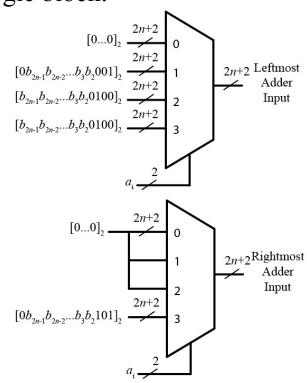


**Figure 3: Combinational Logic Block Detail**

## VI. Implementation and Evaluation

Our methodology utilized the Verilog HDL to capture the quaternary squaring algorithm at the register transfer level for a variety of operand wordsizes. Separate modules were created for the datapath and the controller and these were instantiated in the top-level design through a third Verilog module. The Verilog specifications were then synthesized using the Altera QuartusII design tool suite for programmable logic and the Synopsys Design Compiler for a standard cell ASIC. After synthesis and technology mapping, timing and area analysis tools were used to report on required resources and timing values.

Although our technique allows for $m$ to vary and be specified as a value greater than 2, we chose $m=2$ for the experiments so that we could compare our results to other published approaches. Larger values of $m$ result in higher-performance squaring circuits with proportionally increasing

circuit area. The extreme case of setting $m$ to the entire wordsize of the squarand results in a fully parallel squaring circuit since the residual $r$ becomes equivalent to the overall operand. Alternatively, restricting $m$=1 results in a fully bit-serial architecture. Since [12][13] utilize quaternary recoding, they provide an example equivalent to our case of $m$=2 which is not at one of the two extreme values and can thus be used as a basis of comparison.

Table 3 is provided to summarize past results in squaring methods in order to compare them to the approach described here. The key difference in our approach versus those in Table 3 is that it allows a designer to choose any desired radix, and this choice allows a designer to optimize the resulting latency-area product to better adhere to their specific application. If the squaring circuit is implemented in conventional binary electronic circuitry, the possible radices are restricted to powers of two; however, our approach is general in that it can be applied to future technologies that may utilize any arbitrary radix including those that are not powers of two. Most of the past results utilize binary radices; however, some also employ Booth recoding which can be considered as a special case of radix-4. Only two references use general radix-4 approaches [12][13] whereas our method allows for arbitrarily large radices to be employed. The use of an arbitrary radix is identical to considering our approach as a digit-serial MVL method.

**Table 3: Key Parameters of Cited Squaring Circuits**

| Citation | Architecture | Radix | Notes |
|---|---|---|---|
| [1] | Parallel | 2/Booth | Wall Tree/CSA |
| [3] | Parallel | 2 | Optimized Ling Multiplier |
| [4] | Bit-serial/systolic | 2 | LSb first |
| [5] | Bit-serial with Tree | 2 | LSb first |
| [6] | Bit-serial/systolic with CSA Accumulator | 2 | LSb first |
| [7] | Bit-serial | 2 | LSb first |
| [8] | Parallel | 2 | Improved CSA Tree |
| [9] | Parallel | 2/Booth | Booth Folding |
| [10] | Parallel | 2/Booth | Optimized Version of [8] |
| [11] | Parallel | 2/Booth | Optimized Version of [9] |
| [12] | Parallel | 4 | Approximate Result |
| [13] | Parallel | 4 | Dual-Recoding |
| [17] | Serial/Parallel hybrid | 2 | Parallel on Subwords |
| [18] | Parallel | 2 | Approximate Result; Pipelined |
| [19] | Parallel | 2 | Pipelined & Retimed; Vedic-based |
| [21] | Parallel | 2 | Approximate Result; |
| [22] | Parallel | 2 | Approximate; Table-based |
| [23] | Parallel | 2 | Improved Adder Tree |
| [24] | Parallel/CORDIC Iterative | 2 | Approximate Result; CORDIC |

The squaring algorithm can be implemented in either hardware or as low-level software. We chose to focus on a hardware implementation and consider both FPGA and standard cell ASIC target technologies. The purpose of the ASIC experimental data is to provide a comparison between our approach and a prior art approach [13] for a squaring circuit. We used an older 0.5µm cell library for the experiments. [20] indicates that a more modern 90nm library would increase the frequency by an approximate factor of six (6) and Moore's law indicates that the area would reduce by a factor of $2^{-7}$ assuming a 14-year time span between the availability of 0.5um versus 90µm technology.

For FPGA technology, squaring circuits were synthesized and mapped to three different example Altera FPGA architectures, the 10M02DCU324A7G (MAX 10), the 5SGSMD4E1H29C1 (Stratix V), and 5CEBA2F17A7 (Cyclone V). These are chosen so that the effects of different internal FPGA architectures can be examined to determine how efficiently each can implement the squaring algorithm. Tables 4, 5, and 6 contain a summary of the experimental results for squarand wordsizes of 8, 16, 32, and 64 bits.

**Table 4: Area and Timing using MAX 10 Device**

| Wordsize, *nm* (bits) | # of Comb. Cells | # of Ded. Regs. | FPGA area (%) | Max. Frequency (MHz) | Throughput ($MW/s^{-1}$) |
|---|---|---|---|---|---|
| 8 | 48 | 33 | < 1 | 1328 | 332 |
| 16 | 84 | 62 | < 1 | 1109 | 138 |
| 32 | 168 | 129 | < 1 | 824 | 51.5 |
| 64 | 328 | 257 | < 1 | 638 | 20 |

**Table 5: Area and Timing using Stratix V Device**

| Wordsize, *nm* (bits) | # of Comb. Cells | # of Ded. Regs. | FPGA area (%) | Max. Frequency (MHz) | Throughput ($MW/s^{-1}$) |
|---|---|---|---|---|---|
| 8 | 19 | 33 | 2 | 272 | 67.9 |
| 16 | 30 | 62 | 4 | 206 | 25.8 |
| 32 | 59 | 129 | 7 | 162 | 10.2 |
| 64 | 116 | 258 | 14 | 111 | 3.5 |

**Table 6: Area and Timing using Cyclone V Device**

| Wordsize, *nm* (bits) | # of Comb. Cells | # of Ded. Regs. | FPGA area (%) | Max. Frequency (MHz) | Throughput ($MW/s^{-1}$) |
|---|---|---|---|---|---|
| 8 | 19 | 34 | < 1 | 429 | 107 |
| 16 | 30 | 62 | < 1 | 336 | 42.0 |
| 32 | 60 | 129 | < 1 | 262 | 16.4 |
| 64 | 116 | 257 | 1 | 246 | 7.8 |

We also synthesized the design using a standard cell library available for academic use [16] based on AMI 0.5µm technology. Although 0.5 µm technology is not the most modern feature size available, the point of this experiment is to have a basis for comparison between our algorithm and others when implemented as an ASIC. These comparative results allow one to observe the behavior for our approach versus a prior art fully parallel approach and the area results approximately scale to smaller feature sizes in relation to Moore's law. The clock speed results should increase by approximately a factor of six (6) when compared to a 90nm standard cell library [20]. The results using the standard cell design flow are provided in Table 7.

**Table 7: Area and Timing using 0.5$\mu$m Std Cells**

| Wordsize, *nm* (bits) | # of nets | Area Utilized ($\mu m^2$) | Max. Frequency (MHz) | Throughput ($MW/s^{-1}$) |
|---|---|---|---|---|
| 8 | 94 | 93528 | 158 | 40.0 |
| 16 | 166 | 171504 | 100 | 12.5 |
| 32 | 310 | 328680 | 58 | 3.6 |
| 64 | 598 | 642240 | 31 | 1.0 |

To allow for a comparison in throughput, we also implemented a digit-serial squaring circuit with an architecture based on that given in [17]. The embedded internal multiplication circuit is a quaternary multiplier so that this reference design would have the same characteristics as our test cases for the new methodology. Thus, an equivalent number of iterations are required to produce the final squared result in the reference case as for our approach. Tables 8, 9, and 10 contain the results of the implementation of the multiplier-based quaternary digit-serial circuit. As in Tables 4, 5, and 6 7, the devices used are the 10M02DCU324A7G (MAX 10), the 5SGSMD4E1H29C1 (Stratix V), and the 5CEBA2F17A7 (Cyclone V). Table 11 contains the results of [17] when synthesized using standard cells from [16].

**Table 8: Area and Timing using MAX 10 Device in [17]**

| Wordsize, *nm* (bits) | # of Comb. Cells | # of Ded. Regs. | FPGA area (%) | Max. Frequency (MHz) | Throughput ($MW/s^{-1}$) |
|---|---|---|---|---|---|
| 8 | 48 | 26 | 2 | 226 | 28.3 |
| 16 | 88 | 50 | 4 | 187 | 11.7 |
| 32 | 168 | 98 | 7 | 153 | 4.8 |
| 64 | 328 | 194 | 14 | 118 | 1.8 |

**Table 9: Area and Timing using Stratix V Device in [17]**

| Wordsize, *nm* (bits) | # of Comb. Cells | # of Ded. Regs. | FPGA area (%) | Max. Frequency (MHz) | Throughput ($MW/s^{-1}$) |
|---|---|---|---|---|---|
| 8 | 19 | 26 | < 1 | 608 | 76.0 |
| 16 | 35 | 50 | < 1 | 598 | 37.4 |
| 32 | 67 | 98 | < 1 | 506 | 15.8 |
| 64 | 131 | 194 | < 1 | 374 | 5.8 |

**Table 10: Area and Timing using Cyclone V Device in [17]**

| Wordsize, *nm* (bits) | # of Comb. Cells | # of Ded. Regs. | FPGA area (%) | Max. Frequency (MHz) | Throughput ($MW/s^{-1}$) |
|---|---|---|---|---|---|
| 8 | 19 | 26 | < 1 | 235 | 29.4 |
| 16 | 35 | 50 | < 1 | 206 | 12.9 |
| 32 | 67 | 98 | < 1 | 186 | 5.8 |
| 64 | 132 | 194 | 1 | 153 | 2.4 |

**Table 11: Area and Timing using 0.5$\mu$m Std Cells for [17]**

| Wordsize, *nm* (bits) | # of nets | Area Utilized ($\mu m^2$) | Max. Frequency (MHz) | Throughput ($MW/s^{-1}$) |
|---|---|---|---|---|
| 8 | 110 | 105336 | 165 | 20.6 |
| 16 | 198 | 193752 | 102 | 6.4 |
| 32 | 374 | 370584 | 58 | 1.8 |
| 64 | 726 | 724176 | 30 | 0.5 |

Using a quaternary radix ($\beta$=4) allows our design to generate the square of $\alpha$ in $n$=($nm$)/2 iterations where $nm$ is the wordsize of $\alpha$ in bits. The overall throughput of the circuit is equivalent to $1/[j\tau n]$ in units of words/second where $\tau$ is the period of the clock frequency, $j$ is the number of clock periods required for computation of a single iteration, and each squared result is in units of 'words." Since $nm$ is constant, $n$ decreases in proportion to $m$ and the tradeoff between radix value and the number of required iterations becomes apparent. The other important parameter in the throughput expression is the clock period $\tau$ that is specified to slightly exceed the worst-case register-to-register path delay through the datapath portion of the circuit.

## VII. Conclusions

A digit-serial squaring algorithm based on squarands expressed in any arbitrary radix number system is formulated and implemented in a prototype FPGA logic circuit. The algorithm is motivated by a Vedic technique and is generalized for arbitrary radix values and to account for all possible cases of the value of the squarand. Further motivation for the development of this technique is to allow arithmetic logic circuit designers the ability to trade-off resulting circuit logic resources with throughput by varying the number of bits produced during each iteration through an appropriate choice of the working radix value.

The results of the prototype implementation are analyzed and found to offer a desirable alternative as compared to past squaring circuit approaches where either a bit-serial or fully parallel type of circuit is used. The method is applicable for implementation in software or in hardware. Hardware implementations can be realized in a variety of target technologies including programmable devices, standard cell library ASICs, full-custom ASICs, or any combination of these.

## References


[1] Pihl, J. and Aas, E.J., "A multiplier and squarer generator for high performance DSP applications," *IEEE 39th Midwest Symposium on Circuits and Systems*, vol. 1, pp. 109-112, Aug 1996.

[2] Stallings, W., **Cryptography and Network Security: Principles and Practices**. Prentice-Hall, 4th ed., Upper Saddle River, NJ: 2006.

[3] Chen, T.C., "A Binary Multiplication Based on Squaring" *IEEE Trans. Computers*, vol. C-20, no. 6, pp. 678-80, June 1971.

[4] Pekmestzi, K.Z., Kalivas, P., and Moshopoulos, N., "Long unsigned number systolic serial multipliers and squarers," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 48, no. 3, pp. 316-321, Mar 2001.

[5] Dadda, L., "Squarers for Binary Numbers in Serial Form" in proc. *IEEE Symposium on Computer Arithmetic*, pp. 173-180, June 1985.

[6] Chaniotakis, E., Kalivas, P., and Pekmestzi, K.Z., "Long number bit-serial Squarers," in proc. *IEEE Symposium on Computer Arithmetic*, pp. 29- 36, June 2005.

[7] Ienne, P., and Viredaz, M.A., "Bit-serial multipliers and squarers," *IEEE Transactions on Computers*, vol.43, no.12, pp.1445-1450, December 1994.

[8] Yoo, J.T., Kent, K.F., Smith, F., and Gopalakrishnan, G.G., "A fast parallel squarer based on divide and-conquer," *IEEE J. Solid-State Circuits*, vol. 32, no. 6, pp. 909–912., June 1997.

[9] Strollo, A.G.H. and DeCaro, D., "Booth Folding Encoding for High Performance Squarer Circuits," *IEEE Trans. Circuits and Systems-I1 Analog and Digital Signal Processing*, vol. 50, no. 5, pp. 250-254, May 2003.

[10] Fengqi, Y.Y. and Willson, A.N., "Multirate digital squarer architectures," in proc. *IEEE Int. Conf. on Electronics, Circuits and Systems*, vol. 1, pp. 177-180, Sept. 2001.

[11] DeCaro, D. and Strollo, A.G.M., "Parallel squarer using Booth-folding technique," *Electron. Lett*., vol. 37, no. 6, pp. 346–347, Mar. 2001.

[12] Datla, S.R., Thornton, M.A., and Matula, D.W., "A Low Power High Performance Radix-4 Approximate Squaring Circuit," in proc. *IEEE International Conference on Application-specific Systems, Architectures and Processors*, pp. 91-97, July 2009.
[13] Moore, J., Thornton, M.A., and Matula, D.W., "A Low Power Radix-4 Dual Recoded Integer Squaring Implementation for use in Design of Application Specific Arithmetic Circuits," in proc. *Asilomar Conference on Signals, Systems, and Computers*, pp. 1819-1822, Oct. 2008.
[14] Kornerup, P. and Matula, D.W., **Finite Precision Number Systems and Arithmetic**, Cambridge University Press, Cambridge, United Kingdom, ISBN 978-0-521-76135-2, 2010.
[15] Miller, D.M. and Thornton, M.A., **Multiple-Valued Logic Concepts and Representations**, Morgan & Claypool Publishers, San Rafael, California, ISBN 10-1598291904, 2008.
[16] Stine, J., et al., "FreePDK: An Open-source Variation-aware Design Kit," in proc. *IEEE International Conference on Microelectronic Systems Education* (MSE), pp. 173-174, June 2-4, 2007.
[17] Parhami, B., **Computer Arithmetic: Algorithms and Hardware Designs**, Oxford University Press, New York, New York, ISBN 0-19-512583-5, 2000.
[18] V. Risojevic', A. Avramovic', Z. Babic', and P. Bulic', "A Simple Pipelined Squaring Circuit for DSP," in proc. IEEE *International Conference on Computer Design* (ICCD), pp. 162-167, Oct. 9-12, 2011.
[19] S. Jalaja and A.M.V. Prakash, "High Speed VLSI Architecture for Squaring Algorithm Using Retiming Approach," in proc. *International Conference on Advances in Computing and Communications* (ICACC), pp. 233-238, Aug. 29-31, 2013.
[20] G. Drake, "Dependence on Feature Size," electronic document on the web, Apr. 29, 2011, accessed May 20, 2015, http://psec.uchicago.edu/workshops/fast_timing_conf_2011/system/docs/25/original/110429_psec_workshop_drake_size_dependence.pdf.
[21] Z. Xun, J. Weiwei, and J. Donming, "Circuit Design of an Improved Approximate Squaring Function," in proc. *6th Int. Conf. on ASIC* (ASICON), pp. 1082-1084, 2005.
[22] H.S. Abdel-Aty-Zohdy and A.A. Hiasat, "VLSI Design and Implementation of an Improved Squaring Circuit by Combinational Logic," in proc. *31st Asilomar Conf. on Signals, Systems & Computers* (ASILOMAR), pp. 426-429, 1997.
[23] R.H. Strandberg, J.-C. Le Duc, L.G. Bustamante, V.G. Oklobdzija, and M.A. Soderstrand, "Implementation of Adaptive Rate Kwan-Martin Notch Filter using Efficient Realizations of Reciprocal and Squaring Circuit," in proc. *31st Asilomar Conf. on Signals, Systems & Computers* (ASILOMAR), pp. 324-328, 1994.
[24] S.-F. Hsiao, C.-H. Lee, Y.-C. Cheng, and A. Lee, "Designs of Angle-rotation in Digital Frequency Synthesizer/Mixer using Multi-stage Architectures," in proc. *31st Asilomar Conf. on Signals, Systems & Computers* (ASILOMAR), pp. 2181-2185, 2011.

**Saurabh D. Gupta** received the B.E in electronics engineering at Mumbai University in Mumbai, India in 2009 and the M.S in computer engineering at Southern Methodist University in 2012. He is currently pursuing the Ph.D in computer engineering at Southern Methodist University.

Since 2011, he is a teaching assistant at Southern Methodist University assisting in assembly language programming and machine organization (x86), microprocessor architecture and interfacing (ARM), and digital systems design courses (FPGA). From 2011-2012, he was a research assistant in the area of asynchronous hardware design funded by the National Science Foundation. His research interests include computer arithmetic, electronic design automation for synthesis and verification, embedded systems design, VLSI testing, and hardware Trojan circuit detection.

**Mitchell A. Thornton** (M'85–SM'99) received the B.S. in electrical engineering at Oklahoma State University in Stillwater, Oklahoma in 1985, the M.S. in electrical engineering at the University of Texas-Arlington in 1990 in Arlington, Texas, M.S. in computer science at Southern Methodist University in 1993, and the Ph.D. in computer engineering at Southern Methodist University in 1995.

He was employed at E-Systems, Inc. in Greenville, Texas from 1986 through 1991 and resigned as a Senior Electronic Systems Engineer. He was employed as a Design Engineer at Cyrix Corporation from 1992 through 1993. He has served as a full-time faculty member at the University of Arkansas (1995-1999), Mississippi State University (1999-2002), and is presently a Professor and the Cecil H. Green Chair of Engineering at Southern Methodist University where he is also the Interim Associate Director and Technical Director of the Darwin Deason Institute for Cyber Security. He is a licensed professional engineer in the states of Arkansas, Mississippi, and Texas. Within IEEE, he has served as chair of several committees. His research interests include electronic design automation algorithms for synthesis and verification, computer arithmetic, disaster tolerant systems and modeling, and computer security hardware.