2009 19th IEEE International Symposium on Computer Arithmetic

Higher Radix Squaring Operations Employing Left-to-Right Dual Recoding

David W. Matula
Department of Computer Science and Engineering
Southern Methodist University, Dallas, Texas
matula@lyle.smu.edu

Abstract

We introduce a novel left-to-right leading digit first dual recoding of an operand for the purpose of designing the squaring operation on that operand. Our dual recoding yields an array of non-negative partial squares of size essentially one half that of a comparable multiplier partial product array for both radix-4 and radix-8 designs. For radix-8 design the 128-bit square of a 64-bit operand can be obtained from a consolidated partial square array of just 11 rows. We describe advantages of our left-to-right recoding compared to a previous right-to-left Booth-folding encoding applicable to radix-4. We also show simplifications available to the designs of a rounded floating point square operation and to a low precision approximate square.

Keywords: Booth multiplier recoding, squarer, Booth-folding, partial products, partial squares, sign extension.

1. Introduction and Summary

Squaring is a frequent arithmetic operation with diverse applications. Customized squarers have been discussed for DSPs, e.g. see [8], graphics processors, and reciprocal and transcendental function evaluation, e.g. [1], [18]. Squarers can also be employed to implement multiplication with reasonable efficiency [10], [11], [17].

A considerable number of researchers (e.g. [4], [7], [8], [9], [12], [13]), have contributed to the design of radix-2 squarers exploiting the symmetry $x_i x_j + x_j x_i = 2x_i x_j$ to essentially halve the number of partial products compared to a multiplier. These designs primarily describe hardwired bit arrays for efficient accumulation, mostly focusing on low precision. Since squaring is a unary operation, lookup tables have also been incorporated in proposed squarer designs[17], [19].

Our focus in this paper is on the design of higher radix squaring operations, in particular for radices 4 and 8 where the extensive refinements and efficiencies of Booth multiplier recoding may be incorporated to facilitate the implementation.

For a high radix Booth recoded [15] multiplication, only the multiplier (first operand) is recoded into a digit string in the higher radix while the multiplicand (second operand) remains in binary. Each higher radix recoded multiplier digit multiplies the full multiplicand bit string in a partial product generator, with the array of partial products summed to generate the product.

For a high radix squaring operation, the single operand assumes both the role of the multiplier and multiplicand. Our dual recoding recognizes these distinct asymmetric roles of the single operand. The dual recoding concurrently provides a "squarer" digit string in the high radix and a corresponding sequence of successively truncated "squarands" in binary form. The i^{th} squarer digit multiplies the i^{th} squarand in the i^{th} partial square generator, with the array of partial squares summed to generate the square.

De Caro, Strollo and Napoli [3], [5], [6] introduced a "Booth-folding" encoding for radix-4 demonstrating that the symmetry (folding) and Booth radix-4 recoding features could be combined to obtain a partial square array with essentially only $n^2/4$ bit products, about half the size of a Booth radix-4 recoded partial product array. They effectively utilized the recurrence $(x')^2 = x^2 - (2x' + d)d$ where d is the low order Booth radix-4 digit of x (i.e. $x = x' + d$). This recursion yields a right-to-left low order digit first recoding where each Booth radix-4 digit effectively multiplies bits of greater or equal significance. Importantly, the encoding is shown to require only 1's compliments rather than 2's compliments, but sign extension is still required as in Booth recoded multiplication. Thus the right-to-left Booth-folding encoding is very useful for integer squaring when only the low order p-bits of a $(p \times p)$-bit square are needed [14], but leads to a complex pattern of bit products for

1063-6889/09 $25.00 © 2009 IEEE
DOI 10.1109/ARITH.2009.34

IEEE computer society

the high order part [2]. The Booth folding technique does not extend readily to radix-8, since the determination of the 3× factors for the various low order truncated bit strings are not simply available from bits of a precomputed full precision 3× bit string.

There is a need for a higher radix squaring recoding applicable to both radix-4 and radix-8 that results in partial square generators (PSG's) similar in design to Booth radix-4 and radix-8 recoded PPG's yielding a partial square array of half the size of a comparable multiplier partial product array. For approximate squarers there is a further need for simpler formation of the leading portion of the partial square array from outputs of the PSG's.

In this paper we introduce a left-to-right leading-digit-first squarer dual recoding applicable to both radix-4 and radix-8 recodings.

Our recoding implicitly utilizes the recurrence $(x')^2 = x^2 - d(d + 2x')$, determined from $x^2 = (d + x')^2$, where d is the high order Booth digit of x effectively obtained by rounding-to-nearest with x' the rounded off tail. Here d is a squarer leading digit, $(d + 2x')$ is a squarand, $d(d + 2x')$ is a partial square, and x^2 is obtained as a sum of partial squares involving successively shorter least-significant-digit strings. This should be contrasted with the Booth-folding recurrence $(x')^2 = x^2 - (2x' + d)d$ where $(2x' + d)$ is a squarand, d is a squarer digit, and $(2x' + d)d$ is a partial square with x^2 obtained as a sum of partial squares determined by successively shorter most-significant-digit strings. Our left-to-right leading-digit-first dual recoding provides the following features not obtained by the right-to-left Booth-folding encoding:

- The partial squares are all non-negative, so no sign extensions are needed.
- The partial squares are each scaled down by another power of 16 for radix-4, and 64 for radix-8, yielding convenient formation of the leading half of the partial square array.
- The squarands for digits {±3} are formed by selecting successively shorter low order parts from a precomputed full precision 3× bit string, with modification limited to a few leading bits of the selected parts.
- The sum of k leading (truncated) partial squares from k PSG's can provide an approximate square of accuracy about *4k*-bits for radix-4 and *6k*-bits for radix-8 without need of either sign extension or 2's compliments.

The leading-digit-first dual recoding has the following additional features also obtained (for radix-4) by the Booth-folded encoding:

- The squarer digits are identical to Booth recoded digits for radix-4 and radix-8.
- The squarands are formed by selecting successively shorter bit strings from x with selective shift-compliment for digits {±1, ±2, ±4} as in Booth PPG's.
- The full partial square array can be consolidated to essentially *p*/4 rows for radix-4 and *p*/6 rows for radix-8.

The Booth-folding radix-4 encoding is desirable for generating the low order *p*-bit integer square, as investigated in [14], since the complementation operations are 1's complements [3], [5], [6], and the partial squares are each shifted up by 4-bits, each coming from a single PSG.

Our leading-digit-first dual recoding is more effective for generating high-order parts of a square, such as applicable for approximate squarers. Note that the leading bits of x occur in only a couple partial squares in our dual recoding, whereas they occur in all partial squares of the Booth-folding encoding.

In Section 2 we provide the foundation for our leading-digit-first squaring operand radix-4 dual recoding. Our main results are that the partial squares are all non-negative, that the succession of partial squares are each scaled down by a factor of 16, and that the partial square output by PSG's can be positioned two per row yielding a partial squares array of essentially $n^2/4$ bit products with depth $n/4$.

In Section 3 we extend the foundations to cover radix-8 dual recoding. Our results show the partial squares are all non negative, that the squarer digits are the Booth radix-8 digits {0, ±1, ±2, ±3, ±4}, with the partial squares scaled down by successive factors of 64, which can be output by PSG's two per row yielding a partial square array of essentially $n^2/6$ bit products. Our principle result is that the succession of 3× terms needed as inputs to the *p*/3 PSG's can each be obtained by extracting a low order bit string from a precomputed string for 3× with modification of only four leading bits.

In Section 4 we discuss two limited precision squaring operations where our leading-digit-first dual recoding is particularly effective:

(i) The IEEE standard [16] *p*-bit rounded floating point square of a *p*-bit normalized operand, and

(ii) An *n*-bit fixed point approximate square.

Regarding a rounded square, we introduce condition tests that allow the floating point square to be directly normalized and the exactness flag to be set independent of, and in parallel with, determining the square. Regarding an approximate square operation, we show that low precision approximate squares can be computed by summing just a few partial squares, e.g. we show fixed point 12- (resp. 16-) bit approximate squares can be obtained from the sum of just three (resp. four) partial squares.

2. Radix-4 Left-to-Right Dual Recoding

For the left-to-right leading digit dual recoding, the i^{th} squarand is determined only from bits of lesser or equal significance to the bits determining the i^{th} high radix squarer digit.

The dual recoding for radix-4 is simplest for implementation and will be described in detail first. The dual recoding for radix-8 raises some additional issues for implementation that will be discussed in the next section.

The catalyst for characterizing the left-to-right higher radix dual recoding is the sequence of 2's complement tails of the operand.

Definition 1: Given the p-bit normalized operand $x = 01.b_1b_2 \ldots b_{p-1}$, the radix-4 2's complement tails of x are:

$t_0 = x = 1.b_1b_2 \ldots b_{p-1}$

$t_i = \left(\bar{b}_{2i-1}b_{2i} \,.\, b_{2i+1} \ldots b_{p-1}\right)$

for $1 \le i \le (p+1)/2$, where $\bar{b}_{2i-1} = -b_{2i-1}$. The 2's complement tails are related to Booth radix-4 representation.

Observation 2: $x = \sum_{i=0}^{\lfloor (p+1)/2 \rfloor}(t_i - t_{i+1}/4)4^{-i} = \sum_{i=0}^{\lfloor (p+1)/2 \rfloor} d_i 4^{-i}$, where $d_i = (t_i - t_{i+1}/4)$ is the i^{th} Booth radix-4 digit of x.

Proof: Note that the substring $b_{2i+2}b_{2i+3}\ldots b_{p-1}$ is common to t_i and $t_{i+1}/4$, so that:

$t_i - t_{i+1}/4 = \left(\bar{b}_{2i-1}b_{2i}.b_{2i+1}\right) - \left(0.\bar{b}_{2i+1}\right)$,

so $t_i - t_{i+1}/4 = -2b_{2i-1} + b_{2i} + 2b_{2i+1} \cdot 2^{-1} = d_i$, and $d_i \in \{-2,-1,0,1,2\}$ is recognized as the i^{th} Booth recoded radix-4 digit.

Corollary 2.1: For $0 \le i \le (p+1)/2$, the 2's complement tail is the tail of the Booth radix-4 digit string, i.e.,

$$t_i/4 = 0.d_id_{i+1} \ldots d_{(p+1)/2}\,.$$

The squares of the 2's complement tails are now shown to provide the foundation for our dual operand recoding.

Theorem 3: Let $q_i = t_i + t_{i+1}/4$, for $0 \le i \le (p+1)/2$. Then

$x^2 = \sum_{i=0}^{\lfloor (p+1)/2 \rfloor} d_iq_i16^{-i}$

Furthermore, when d_i and q_i are non zero, they both have the same sign $(-1)^{b_{2i-1}}$, so then:

$x^2 = \sum_{i=0}^{\lfloor (p+1)/2 \rfloor} |d_i||q_i|16^{-i}$.

Proof: Since $x^2 = \sum_{i=0}^{\lfloor (p+1)/2 \rfloor}(t_i^2 - t_{i+1}^2/16)16^{-i}$, then $(t_i^2 - t_{i+1}^2/16) = (t_i - t_{i+1}/4)(t_i + t_{i+1}/4) = d_iq_i$. It is readily seen that: $d_i = (-1)^{b_{2i-1}}|d_i|$, and $q_i = (-1)^{b_{2i-1}}|q_i|$,

so $d_iq_i = |d_i||q_i|$ for $0 \le i \le \lfloor (p+1)/2 \rfloor$. □

Figure 1 illustrates the cancellation (and deletion) of bit b_{2i+1} and left shift of bits $b_{2i+2}b_{2i+3} \ldots b_{p-1}$ in forming the bit string for q_i resulting in:

$q_i = (t_i + t_{i+1}/4) = \bar{b}_{2i-1}b_{2i}.b_{2i+2}b_{2i+3} \ldots b_{p-1}$.

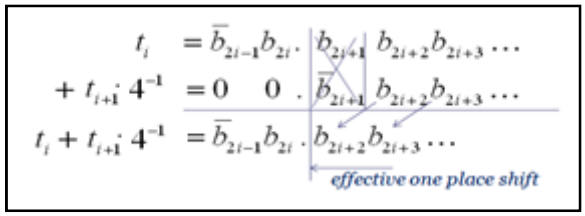

Figure 1: $(t_i + t_{i+1}\cdot 4^{-1})$ **yields** $q_i = \bar{b}_{2i-1}b_{2i}.b_{2i+2}b_{2i+3}\ldots$.

By performing the 2's complement when dictated by $b_{2i-1} = 1$ and deleting the resulting sign, we obtain:

$$|q_i| = \begin{cases} b_{2i}.b_{2i+2}b_{2i+3} \ldots b_{p-1} & \text{for } b_{2i-1} = 0, \\ b'_{2i}.b'_{2i+2}b'_{2i+3} \ldots b'_{p-2}b''_{p-1} & \text{for } b_{2i-1} = 1, \end{cases}$$

where $b'_j = (1 - b_j)$ for $1 \le j \le p-2$, and $b''_{p-1} = (2 - b_{p-1})$.

In summary then our radix-4 dual recoding for squaring provides the sequence $d_0, d_1, \ldots, d_{(p-1)/2}$ of squarer digits where $d_i = -2b_{2i-1} + b_{2i} + b_{2i+1}$ is the i^{th} radix-4 Booth recoded digit for $0 \le i \le (p-1)/2$. The dual recoding concurrently provides the sequence $q_0, q_1, \ldots, q_{(p-1)/2}$ of squarands, with $d_iq_i16^{-i} = |d_i||q_i|16^{-i}$ the i^{th} partial square for $0 \le i \le (p-1)/2$. □

The radix-4 dual recoding for left-to-right squaring has a number of properties of considerable practical value for applications:

- The partial squares $|d_i||q_i|16^{-i}$ are all non-negative, so no sign extensions are needed.
- The partial squares are each scaled down by another power of 16, so an n-term sum

provides an approximate square of about $4n$ bits of accuracy.

The partial square generators are similar in design to Booth radix-4 partial product generators but simpler in two ways – no sign extensions are needed, and, on average, they are about half the size for the same precision.

2.1 2*p*-bit Partial Square Arrays

For a p-bit normalized operand $x = 01.b_1b_2 \ldots b_{p-1}$, the radix four dual recoding yields a $\lfloor p/2 \rfloor + 1$ term sum where the partial squares are always non-negative but may be subject to a 2's complement operation in their formation.

Note that when p is even, the $p/2^{\text{th}}$ tail is $t_{p/2-1} = \bar{b}_{p-3}b_{p-2}b_{p-1}$ and the square satisfies $(t_{p/2-1})^2 \in \{0, 1, 4, 9, 16\}$ with the low order two bits of $(t_{p/2-1})^2$ given by $0b_{p-1}$. Then letting $(t_{p/2-1})^2 = a_4a_3a_20b_{p-1}$, it is clear that $(t_{p/2-1})^2$ can be taken as the final partial square with a "hardwired" result avoiding any 2's complementation.

When p is odd, the final term is $t_{(p-1)/2} = \bar{b}_{p-2}b_{p-1}$ with $(t_{(p-1)/2})^2 \in \{0,1,4\}$ and can similarly be hardwired, so we obtain the following.

Observation 4: The full precision *2p*-bit square may be formed as the sum of $\lceil p/2 \rceil$ partial squares, where the first and last terms are not subject to any conditional complementation.

Example: Consider the normalized 16-bit operand $x = 01.10\ 00\ 10\ 11\ 00\ 01\ 011$. Figure 2 illustrates the 8 selection digits and the partial square array, where the final partial square is the modified 3-bit tail square. Since the low order bit of x has $b_{p-1} = 1$, we have simplified the 2's complement in this example by employing $b''_{p-1} = 1$.

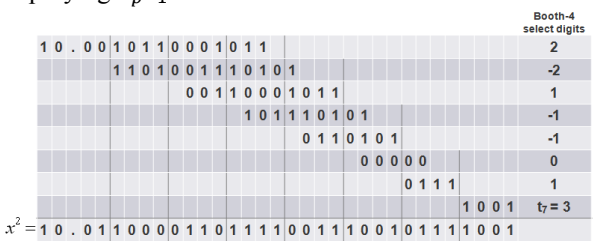

Figure 2: The partial square array for the 32-bit square of the 16-bit normalized operand $x = 01.10\ 00\ 10\ 11\ 00\ 01\ 011$ employing the left-to-right radix-4 leading digit dual recoding.

The partial square array of Figure 2 can be usefully compared with the partial square array for Booth-folding radix-4 recoding given in Figure 3.

Not that corresponding longest-to-shortest partial squares are essentially 2 bits shorter in width for left-to-right dual recoding facilitating reduction to $p/4$ rows. The Booth-folding right-to-left recoding has the advantage of not needing 2's complement bits, but sign extension is needed corresponding to negative valued Booth recoded digit values.

$(d_7d_6d_5d_4d_3d_2d_1d_0)_4 = (\bar{1}012\ \bar{2}1\bar{1}\bar{1})_4$ 1100 0101 1000 1011	Selection Digits
10 0111 0100 1110 10o1	$\bar{1}$
1001 1101 0011 10o1	$\bar{1}$
01 1000 1011 00o1	1
0011 1010 01o0	$\bar{2}$
11 0001 01o0	2
0110 00o1	1
00 00o0	0
10o1	$\bar{1}$
-(o0o0 o0o1 o0o1 o1)	2's comp bits (negative)
1001 1000 0110 1111 0011 1001 0111 1001 (9 8 6 15 3 9 7 9)16	Square 2,557,426,041

Figure 3: The partial square array for the 32-bit square of the 16-bit integer operand $x = 1100\ 0101\ 1000\ 1011$ employing Booth-folding radix-4 right-to-left recoding.

The left-to-right leading digit dual recoding needs no sign extensions but in the general case must deal with 2's complement low order bit implementation issues. When the last unit bit position is not known it is possible to handle all the 2's complement bits by an extra low order row. Let c_i denote the extra 2's complement bit for the i^{th} partial square where $c_i = 1$ if $d_i > 0$, otherwise $c_i = 0$. No complement bit is needed for the initial partial square since $d_0 > 1$ for our dual operand recoding, and none is needed for the last partial square when $t_{p/2-1}$ is determined directly. The partial squares may be consolidated into $n = \lceil p/4 \rceil$ rows and one exceptional bit position as illustrated in Figure 4 for p=16.

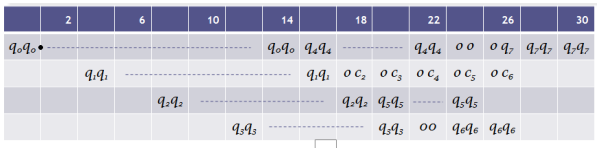

Figure 4: Consolidated 4-row (plus exceptional bit) partial square array for the 32-bit square of a 16-bit normalized operand employing left-to-right radix-4 leading digit dual recoding.

The bit positions labeled q_i and c_i in Figure 4 denote positions allocated to the partial square $|d_i||q_i|$. The consolidated array is indicative of the general case for $p = 4n$, where the resulting n row array has all but c_1 positioned in the second row after the partial square $|d_1||q_1|$. The extra bit c_1 must be forced in to the

corresponding column as an $(n+1)^{th}$ entry for the case $p = 4n$.

When $p = 4n - j$ for $1 \le j \le 3$, the bit c_1 may be absorbed into the n rows by an appropriate complementation procedure, as summarized in the following.

Observation 5: The 2p-bit square of a normalized p-bit operand $x = 01.b_1b_2 \ldots b_{p-1}$ may be realized as the sum of a consolidated $n = \lceil p/4 \rceil$ row array of partial squares where an additional single bit entry is needed forming one $(n+1)$-bit column when $p = 4n$.

The full double precision result is most useful for support of multiple-precision arithmetic, where a squaring unit of relatively large precision would be most desirable. For very large precisions it is worth considering the radix-8 recoding.

3. Radix-8 Left-to-Right Dual Recoding

For the radix-8 left-to-right leading digit dual recoding (radix-8 dual recoding) the development is similar to the radix-4 dual recoding here yielding squarer (Booth radix-8) digits {0, ±1, ±2, ±3, ±4} and squarands q_i where each partial square $d_iq_i64^{-i}$ is obtained from bits of lesser or equal significance to the bits determining q_i. The development is summarized up to the critical point where we focus in more detail on the issue of how to efficiently obtain the partial squares $d_iq_i64^{-i}$ whenever $|d_i| = 3$.

The dual radix-8 recoding derives from the recurrence $(x')^2 = x^2 - d(d + 2x')$ where d is a leading radix-8 Booth digit of x and x' is the resulting rounded off tail. The foundations derive readily from focusing on the sequence of rounded off 2's complement format tails by the same arguments employed for the radix-4 recoding of Section 2.

Definition 6: Given the p-bit normalized operand $x = 1b_1.b_2b_3 \ldots b_{p-1}$, the radix-8 2's complement tails of x are

$t_0 = x = 1b_1.b_2b_3 \ldots b_{p-1}$

$t_i = \overline{b}_{3i-1}b_{3i}b_{3i+1}.b_{3i+2} \ldots b_{p-1}$,

for $1 \le i \le p/3$, where $\overline{b}_{3i-1} = -b_{3i-1}$.

Definition 7: For $i = 0,1, \ldots, \lfloor p/3 \rfloor$,

$d_i = t_i - t_{i+1}/8$ is the i^{th} radix-8 squarer digit,

$q_i = t_i + t_{i+1}/8$ is the i^{th} radix-8 squarand,

$d_iq_i64^{-i}$ is the i^{th} partial square.

Observation 8: $x^2 = \sum_{i=0}^{j} d_i\, q_i 64^{-i} + t_{j+1}^2$ for $0 \le j \le p/3 - 1$.

Observation 9: $x^2 = \sum_{i=0}^{\lfloor p/3 \rfloor} d_iq_i64^{-i}$.

Observation10: For $i = 0,1, \ldots, \lfloor p/3 \rfloor$,

(i) $d_i = -4b_{3i-1} + 2b_{3i} + b_{3i+1} + b_{3i+2}$, where $-4 \le d_i \le 4$ is the i^{th} Booth radix-8 digit,

(ii) $q_i = \overline{b}_{3i-1}b_{3i}b_{3i+1}.b_{3i+3}b_{3i+4} \ldots b_{p-1}$

(iii) $d_iq_i = |d_i||q_i|$.

As in the radix-4 dual recoding the bit b_{3i+2} contributes as a carry-in to the formation of the squarer digit d_i, but it cancels out in the formation of q_i. The formation of the digit multiples d_iq_i for $d_i \in \{0, \pm1, \pm2, \pm4\}$ are readily obtained in a radix-8 PSG by conditional shift and complement of the selected subsequence (deleting b_{3i+2}) $b_{3i}b_{3i+1}b_{3i+3}b_{3i+4} \ldots b_{p-1}$. No sign extensions are needed. To complete the radix-8 PSG design we shall show how to determine $3q_i$ for $i = 0,1, \ldots, \lfloor p/3 \rfloor$ from a precomputed full precision value of $3x$ with modifications limited to four leading bits of the selected substring. Let

$x = 1b_1.b_2b_3 \ldots b_{p-1}$,

$3x = a_{-2}a_{-1}a_0a_1.a_2 \ldots a_{p-1}$.

Note that $3q_i$ may be written as the sum of an integer k_i with $-12 \le k_i \le 11$ and a fraction $x = 0.f_{i,3i+3}f_{i,3i+4}f_{i,3i+5} \ldots f_{i,p-1}$.

Lemma 11: With $3q_i = k_i + f_i$, for $i = 0,1, \ldots, \lfloor p/3 \rfloor$, f_i is determined from $3x = a_{-2}a_{-1}a_0a_1.a_2 \ldots a_{p-1}$ by $f_i = 0.a_{3i+3}a_{3i+4} \ldots a_{p-1}$.

Proof: The fractional part of $3q_i$ is the fractional part of $3(0.b_{3i+3}b_{3i+4} \ldots b_{p-1})$, which is the same as the fractional part of $2^{3i+1}(3x) = a_{-2}a_{-1} \ldots a_{3i+2} + 0.a_{3i+3}a_{3i+4} \ldots a_{p-1}$. □

Whenever $\overline{b}_{3i-1}b_{3i}b_{3i+1}b_{3i+2}$ yields $|d_i| = 3$, we need to show how to determine k_i from the four bit string $a_{3i-1}a_{3i}a_{3i+1}a_{3i+2}$. The strings 0101 and 0110 yield $d_i = 3$, with strings 1001 and 1010 yielding $d_i = -3$.

Lemma 12: Let $3q_i = k_i + f_i$ for $i = 0,1, \ldots, \lfloor p/3 \rfloor$, with $d_i = 3$. Then k_i is determined from $a_{3i-1}a_{3i}a_{3i+1}a_{3i+2}$ by the modular integer sum

$k_i = |a_{3i-1}a_{3i}a_{3i+1}a_{3i+2} + 0111|_{16}$

Proof: There are two cases where $d_i = 3$. For the first case, suppose $\overline{b}_{3i-1}b_{3i}b_{3i+1}b_{3i+2} = 010.1$. Then

$9 + 3q_i = 9 + 3(010.) + 3\left(b_{3i+3}b_{3i+4} \ldots b_{p-1}\right)$

$$= 3(0101.b_{3i+3}b_{3i+4}\dots b_{p-1})$$
$$= 3(b_{3i-1}b_{3i}b_{3i+1}b_{3i+2}.b_{3i+3}\dots b_{p-1})$$
$$= 16c + (a_{3i-1}a_{3i}a_{3i+1}a_{3i+2}.) + f_i$$

for some integer c . Then

$3q_i = 16c^* + (a_{3i-1}a_{3i}a_{3i+1}a_{3i+2}.+0111.) + f_i$

for some integer c^*, and the result holds in this case.

For the second case, suppose $\overline{b}_{3i-1}b_{3i}b_{3i+1}b_{3i+2} = 011.0$.

$$9 + 3q_i = 9 + 3(011.) + 3(b_{3i+3}b_{3i+4}\dots b_{p-1})$$
$$= 3(0110.b_{3i+3}b_{3i+4}\dots b_{p-1})$$
$$= 16c + (a_{3i-1}a_{3i}a_{3i+1}a_{3i+2}.) + f_i,$$

so the result holds for both cases yielding $d = 3$. □

Lemma 13: Let $3q_i = k_i + f_i$, for $i = 0,1,\dots,\lfloor p/3$, with $di=-3$. Then ki is determined from $a_{3i-1}a_{3i}a_{3i+1}a_{3i+2}$ by

$k_i = |a_{3i-1}a_{3i}a_{3i+1}a_{3i+2}.+1011|_{16} - 16.$

We note without proof that Lemma 13 follows similarly to the proof of Lemma 12 for both bit strings $\overline{b}_{3i-1}b_{3i}b_{3i+1}b_{3i+2} = \overline{1}00.1$, and $\overline{1}01.0$, where $d_i = -3$.

The results of Lemmas 11-13 can be combined to provide a convenient representation of $3q_i$, corresponding to all four cases where $|d_i| = 3$ by employing the "sign bit" b_{3i-1} in a controlling role. Let $b'_{3i-1} = 1 - b_{3i-1}$. Note then that for $b_{3i-1} = 1$, we can represent integer 9 by the bit string $b_{3i-1}b'_{3i-1}b'_{3i-1}1$, and for $b_{3i-1} = 0$ we obtain $b_{3i-1}b'_{3i-1}b'_{3i-1}1 = 7$.

Theorem 14: For radix-8 dual recoding when $|d_i| = 3$, the value of $3q_i$ can be determined from the substring $a_{3i-1}a_{3i}a_{3i+1}a_{3i+2}$ of $3x = a_{-2}a_{-1}a_0a_1.a_2\dots a_{p-1}$ and the single "sign bit" b_{3i-1} by $3q_i = \overline{b}_{3i-1}a^*_{3i-1}a^*_{3i}a^*_{3i+1}a^*_{3i+2}.a_{3i+3}\dots a_{p-1}$ with $a^*_{3i-1}a^*_{3i}a^*_{3i+1}a^*_{3i+2} = |a_{3i-1}a_{3i}a_{3i+1}a_{3i+2} + b_{3i-1}b'_{3i-1}b'_{3i-1}1|$.

Example: Consider the normalized 16-bit operand $x = 011.000\ 101\ 100\ 010\ 11$, which is the same bit sequence of the example of Section 2, here normalized for radix-8 dual recoding.

$$3x = a_{-2}a_{-1}a_0a_1.a_2a_3\dots a_{15}$$
$$3x = 1001.01000010100001$$
$$x = d_0.d_1d_2\dots = 3.1\overline{24}3\overline{2}_8.$$

Corresponding to $d_0 = 3$, $f_0 = 0.a_3a_4\dots a_{15}$, and $3q_0 = k_0 + f_0 = k_0 + (.100\ 001\ 010\ 000\ 1)$. From Theorem 14, $k_0 = |0010 + 0111|_{16} = 1001$, and $3q_0 = 1001.100\ 001\ 010\ 000\ 1$.

Corresponding to $d_4 = 3$, $f_4 = 0.a_{15}$, and

$$3q_4 = k_4 + f_4 = k_4 + (.1),$$
$$k_4 = |0000 + 0111|_{16} = 0111,$$
$$3q_4 = 0111.1\ .$$

The full 32-bit array is shown in Figure 5.

To practically test the radix-8 recoding, we implemented a software version employing the results of observations 8-10 and Lemmas 11-13. The software implementation of the radix-8 squarer was exhaustively applied to all 2^{23} single precision operands ($p = 24$) over the standard binade $[1,2)$, and the resulting value for the square agreed with the output of a double precision multiplier in all cases.

Note in Figure 5 that the successive partial squares for radix-8 left-to-right recoding recede by 6 bit positions from the left and are tapered to each have 3 bits less width. The 6-row array is seen to be readily consolidated into a 3-row array, which could be designed to absorb 2's complement bits.

Consider that a radix-8 dual operand recoding could be used to provide a $\lceil 129/6 \rceil = 22$ row consolidated partial square array for obtaining the 256-bit square of a 128-bit operand. This is some 10 rows less than the $\lceil 128/4 \rceil = 32$ row consolidated partial square array for the 256-bit square using a radix-4 dual recoding, and 21 rows less than the 129/3 = 43 row partial product array for a Booth radix-8 recoded multiplier.

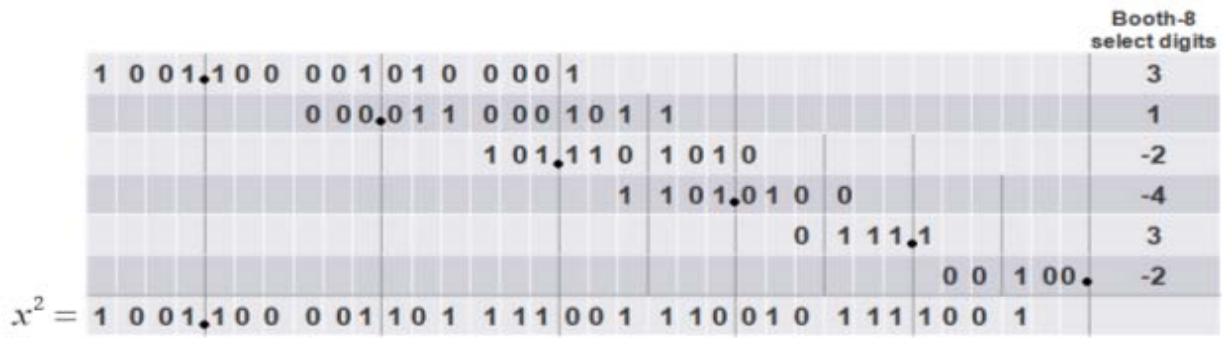

Figure 5: The partial square array for the 32-bit square of $x = 011.000\ 101\ 100\ 010\ 110$ employing the radix-8 leading digit dual recoding.

4. Applications and Implementation Efficiency

There are three fundamentally distinct computational environments where the square of a normalized p-bit operand $x = 1.b_1b_2 \ldots b_{p-1}$ may be implemented with further application specific architectural enhancements.

- $2p$-bit full precision square: this is the full exact double precision square. This result is useful for support of multiple-precision arithmetic.
- p-bit rounded floating point squares: these are the precisely rounded floating point results with particular reference to IEEE standard precisions 24, 53, and 64 corresponding to the single, double, and double extended formats prevalent in commodity microprocessors.
- n-bit fixed point approximate squares: these are $(n+g)$-bit fixed point squares typically accurate to one or two units in the n^{th} place where g is the number of guard bits, typically with $0 \le g \le 3$.

We have previously described the $2p$-bit full precision square. Here we add results pertaining to the precisely rounded and approximate squares.

4.1 IEEE Standard Floating Point Square

For selectable precise floating point roundings a first intermediate result of the product of two normalized p-bit operands, x and y, is the $(p+3)$-bit value

$$z(x,y) = i_1 i_0 \,.\, a_1 a_2 \ldots a_{p-2}\; g\; r\; s,$$

where the leading $(p+2)$-bits of $z(x,y)$ are the leading $p+2$ bits of the fixed point $2p$-bit product $x \times y$, and s is the OR function of the low order $(p-2)$-bits of $x \times y$ with $s = 0$ only if these $p-2$ bits are all zero, indicating $z(x,y) = x \times y$ is the exact product.

After the bits $\{i_1, g, r, s\}$ are found, typically derived from the full $2p$-bit product $x \times y$, a second intermediate result $z'(x,y) = 1\,.\,a'_1 a'_2 \ldots a'_{p-1}\, r'\, s'$ is determined by conditionally shifting right one position when $i_1 = 1$, with $s' = r$ OR s.

For our preceding $p = 16$ bit example $x =$ 01.1000 1011 0001 011, we would obtain $z(x,x) =$ 10.01 1000 0110 1111 001

with $i_1 = 1$ and $grs = 001$, and then $z'(x,x) =$ 1.0011 0000 1101 1110 1 with $r's' = 01$.

There are several independent concurrent computations that can be made for determining some of all of the bits $\{i_1, g, r, s\}$ for the square $z(x,x)$ that allow us to directly determine the corresponding exponent for the normalized intermediate result $z'(x,x) = 1\,.\,a'_1 a'_2 \ldots a'_{p-1}\, r'\, s'$ and whether or not $z'(x,x)$ is exactly equal to x^2.

Observation 15: For the squaring approximation $z(x,x) = i_1 i_0 \,.\, a_1 a_2 \ldots a_{p-2}\; g\; r\; s$, we obtain $i_1 = 1$ if and only if $x > \sqrt{2}$.

Since $\sqrt{2}$ is irrational, it is then sufficient to check if x is greater than the leading p bits of $\sqrt{2}$ while the partial squares are being accumulated so the sum may be generated in its normalized form with the corresponding exponent and the position for the round bit *r'* uniquely identified.

Definition 16: Let the significant *width* $w(x)$ of the normalized p-bit operand $x = 1.b_1b_2 \ldots b_p$ be determined by $x = 1.b_1b_2 \ldots b_{w-2}100 \ldots 0$, where $1 \le w \le p$ with x having exactly p-w low order zeros.

Thus we may refer to $x = 1.b_1b_2 \ldots b_{w-2}1$ as doubly normalized of width w when the width is known. Note then that x^2 has width $2w$ for $x > \sqrt{2}$, and $2w$-1 otherwise. It follows that knowledge of the p-bit normalized operand x is sufficient to determine if the normalized rounded p-bit floating point square is exact except for the particular case where p is odd and $w(x) = (p+1)/2$, as summarized in the following.

Observation 17: Let x be a normalized p-bit operand of width $w(x)$. Then

(i) for $w \ge \lceil p/2 \rceil + 1$, the intermediate result $z'(x,x)$ has $s' = 1$ and x^2 must be rounded to an inexact p-bit floating point square,

(ii) for $w \le \lceil p/2 \rceil, x^2$ is exact of width $w(x^2) \le p$,

(iii) for $w = \lceil p/2 \rceil$ with p odd,

  a. x^2 is exact of width $w(x^2) = p$, for $x < \sqrt{2}$,

  b. x^2 must be rounded to an inexact p-bit floating point square since $w(x^2) = p + 1$, for $x > \sqrt{2}$.

Thus for IEEE standard single (p = 24) or double extended (p = 64) computation of x^2, knowledge of the width $w(x)$ of the operand x is sufficient to set the required exactness flag. For IEEE standard double precision (p = 53), it is further sufficient to check

whether $x < \sqrt{2}$ only for the particular case that $w(x) = 27$.

A further simplification when w is known is that the 2's complement may be formed as a 1's complement with the $(p - w + 1)$ low order bits forced to 100…0, simplifying the partial square array as illustrated in Figures 3 and 6 and avoiding the need for the extra 2's complement bit positions shown in Figure 5 for the case $p = 4n$.

For our preceding 16-bit example with $x =$ 01 .10 0010 1100 0101 1, we obtain w=16. Thus s'=1 and $z'(x,x) \neq x^2$, indicating the rounded result will be inexact by any IEEE rounding mode. Furthermore, since $x > \sqrt{2}$, the leading 17 bits of x^2 may be routed to the lead portion of $z'(x,x)$, avoiding the normalization step prior to rounding needed in typical multiply implementations.

Determination of the leading normalized $(p + 1)$ bits of x^2 through the round bit r still requires accumulation from the low order positions. The nature of the contributions to the lowest four to six bits of the $2p$-bit square by the arrays in Figures 3, 5, and 6 show that simplifications are possible here also. In general, whether or not w is known, the last partial square of the $2p$-bit product is not needed. When the width w is known, the consolidation shown in Figure 4 may be simplified to that shown in Figure 6, where the lowest six bits are not needed for the accumulation to obtain the round bit.

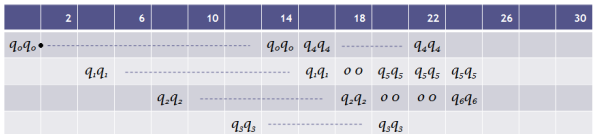

Figure 6: Truncated consolidated radix-4 recoded 4 row partial square array sufficient to obtain the leading 18 bits through the round bit for the square of a 16-bit normalized operand when the width is known, simplifying the 2's complement operation on the squarands.

More generally, when the sticky bit s' and position for r' are known for the $2p$-bit product x^2, alternative logic may be provided to simply provide net carry out information without the need to identify the lowest j bits for any fixed j, $1 \leq j \leq p - 2$.

4.2 $(n + g)$-bit fixed point approximate square

For the approximate square let the operand be a p-bit fraction $x = 0.b_1b_2 \ldots b_p$, where x is normalized when $b_1 = 1$. It is often sufficient for specific applications to obtain an n-bit approximate square, where n is considerably smaller than p.

Definition 18: Given the p-bit operand $x = 0.b_1b_2 \ldots b_p$, then $a_n(x^2) = 0.a_1a_2 \ldots a_na_{n+1} \ldots a_{n+g}$ is an f-ulp $(n + g)$-bit approximate square if $|x^2 - a_n(x^2)| \leq f \cdot 2^{-n}$, where $\frac{1}{2} \leq f \leq 2$ is the number of units in the last (i.e. n^{th}) place (ulps), and g is the number of guard bits, where typically $0 \leq g \leq 3$. The approximation is strict when the ulp bound is strict, and may be directed or undirected.

We are primarily interested in directed f-ulp approximations for $f < 2$, and approximations of at most one ulp for the undirected case. Note that the one ulp bound may be obtained from the 2-ulp directed bound by adding a unit in the summation process.

Observation 19: The leading digit radix-4 dual operand recoding of the fixed point operand $x = 0.b_1b_2 \ldots b_p$ allows us to obtain a $(4n + 3)$-bit strict 2 ulp directed $(4n + 3)$-bit approximate square as the sum of n truncated approximate squares, for $n \leq 7$.

Proof Outline:

Let $t_i = \bar{b}_{2i}.b_{2i+1} \ldots b_p$. Proceeding as in Theorem 3, we note $x^2 = (\sum_{i=0}^{n-1} d_iq_i16^{-i}) + t_n^2 16^{-n}$, where $t_i^2 \leq 1$, and d_i is obtained from $\bar{b}_{2i}.b_{2i+1}b_{2i+2}$ normalized so that $0 \leq d_iq_i \leq 1$. When $d_i < 0$, the 1's complement through bit $4n + g$ is employed. The sum $\sum_{i=0}^{n-1} d_iq_i16^{-i}$ with terms truncated at position $(4n + 3)$ is the desired lower bound.

For our example case $x =$ 01.1000 1011 0001 011, Figure 7 illustrates a 4-partial square array sufficient to yield a 16-bit 1½ ulp lower bound on x^2 employing $g = 3$ guard bits. Note the use of the 1's complement for the partial square for the digit -2.

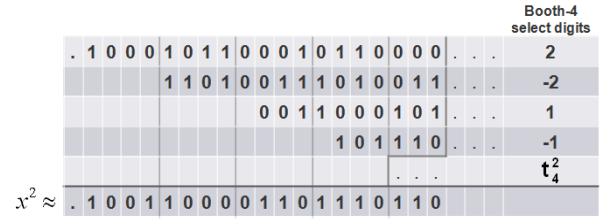

Figure 7: Fixed point 16-bit 1½ ulp lower bound on x^2 from Booth radix-4 four digit sum.

The facility to generate approximate squares of $4n$-bit accuracy from an n-term sum for small n (e.g. $3 \leq n \leq 7$ yielding 12-28 bit approximations) should be of considerable value for various applications from multimedia computations to internal ALU micro-coded division and transcendental algorithms.

5. Conclusions and Further Directions

This paper presented the foundations for a high radix left-to-right leading digit dual recoding of the operand x for computing the unary operation x^2. The method is applicable to both radices 4 and 8, with one part of the dual recoding yielding a squarer digit string identical to the Booth radix-4 and 8 multiplier digit string. The novel additional part of the dual recoding is a sequence of successively truncated terms called squarands, generalizing the notion of the multiplicand as employed in a high radix multiplier design. All partial squares are shown to be non negative, avoiding the complexity of sign extensions. Furthermore, the resulting array of partial squares is essentially just half the size of a comparable partial product array. The dual recoding was shown to have advantages over the right-to-left Booth-folding encoding which is practically limited to radix-4.

The dual recoding procedure was analyzed for application to three types of squaring operations: the full double length square, the single length rounded floating point square, and a small precision approximate square. The approximate square determination is a key component for a new multiplicative division algorithm focused on the family of single, double, and double extended precision IEEE standard floating point division operations that will be the subject of future work of this project.

Synthesis results for a radix-4 dual recoded approximate squarer are given in [20], and further synthesis studies are in progress in our lab.

Acknowledgement: I wish to thank Ilteris M. Derici and Jonathan Y. Zhang for software implementation and testing of the radix-4 and radix-8 dual recoding squaring procedures.

References

[1] P. M. Farmwald, "High Bandwidth Evaluation of Elementary Functions," *in Proc. IEEE 5th Symp. Computer Arithmetic*, pp. 139-142. 1981.

[2] M. Ercegovac, "Left-to-Right Squarer with Overlapped LS and MS parts", *Conference Record of the 37th Asilomar Conference on Signals, Systems and Computers*, vol. 2, pp.1451-1455. November 2003.

[3] A.G.H. Strollo and D. De Caro, "Booth Folding Encoding for High Performance Squarer Circuits" *IEEE Trans. Circuits and Systems-I1 Analog and Digital Signal Processing,* 50(5):250-254, 2003.

[4] Y.Yu Fengqi and A. N.Willson,"Multirate digital squarer architectures," in *Proc. 8th IEEE Int. Conf. on Electronics, Circuits and Systems (ICECS 2001)*, Malta, Sept. 2–5, 2001, pp. 177–180.

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

[6] A. G. M. Strollo, E. Napoli, and D. De Caro, "New design of squarer circuits using Booth encoding and Folding techniques," in *Proc. 8th IEEE Int. Conf. on Electronics, Circuits and Systems (ICECS 2001)*, 2001, pp. 193–196.

[7] R. K. Kolagotla and W. R. Griescbach, "VLSI implementation of a 350 MHz 0.35 m 8 bit merged squarer," *Electron. Lett.*, vol. 34, no. 1, pp.47–48, 1998.

[8] J. Pihl and E. J. Aas, "A multiplier and squarer generator for high performance DSP applications," in *Proc. IEEE 39th Midwest Symp. on Circuits and Systems*, 1996, pp. 109–112.

[9] J.-T. Jae-tack Yoo, K. F. Kent F. Smith, and G. Ganesh Gopalakrishnan, "A fast parallel squarer based on divide-and-conquer," *IEEE J. Solid-State Circuits*, vol. 32, pp. 909–912, 1997.

[10] H.Ling,"High-speed computer multiplication using a multiple-bit decoding algorithm," *IEEE Trans. Computers*, C-19, Aug. 1970, pp. 706-709.

[11] T.C. Chen, "A Binary Multiplication Based on Squaring." *IEEE Trans. Computers*, C-20:678-80, 1971.

[12] K.E. Wires, M.J. Schulte, L.P. Marquette: and P.I. Balzola. "Combined Unsigned and Two's Complement Squarers", *Conference Record of the 31st Asilomar Conference on Signals, Systems and Computers*, V01.2: 1215-1219, 1999.

[13] L. Dadda, "Squarers for binary numbers in serial form", in *Proc. IEEE 7th Symp. Computer Arithmetic*, 1985.

[14] J. Moore, D.W. Matula, M.L. Thornton "A Low Power Radix-4 Dual Recoded Integer Squaring Implementation For Use in Design of Application Specific Arithmetic Circuits", *Asilomar Conference on Signals, Systems and Computers,* October 2008.

[15] L.P. Rubinfeld, "A Proof of the Modified Booth's Algorithm for Multiplication", *IEEE Trans. Computers*, vol. 24, no. 10, pp. 1014-1015 Oct. 1975.

[16] IEEE Standard for Floating-point Arithmetic, IEEE Standard 754-2008, New York, IEEE, August 2008.

[17] C.L. Wey and M.D. Shieh. "Design of a High-Speed Square Generator", *IEEE Trans. Computers*, vol. 47, no. 9, pp. 1021-1026, 1998.

[18] A. Liddicoat and M. J. Flynn, "Parallel Square and Cube Computations", *Proc.* 34th *Asilomar Conference on Signals, Systems & Computers*, 2000.

[19] E.G. Walters III, J.S. Schlessman, and M.J. Schulte, "Combined Unsigned and Two's Complement Hybrid Squarers", IEEE 2001.

[20] S.R. Datla, M.A. Thornton, D.W. Matula, "A Low Power High Performance Radix-4 Approximate Squaring Circuit", submitted for publication.