A new recursive multibit recoding algorithm for high-speed an low-power multiplier.

Abdelkrim K. Oudjida, Nicolas Chaillet, Ahmed Liacha, Mohamed L. Berrandjia

A New Recursive Multibit Recoding Algorithm for High-Speed and Low-Power Multiplier

A.K. Oudjida[1], N. Chaillet[2], A. Liacha[1], and M.L. Berrandjia[1]

(1) Centre de Développement des Technologies Avancées, Algiers, Algeria
(2) Institut FEMTO-ST, Besançon, France

***Abstract*—In this paper, a new recursive multibit recoding multiplication algorithm is introduced. It provides a general space-time partitioning of the multiplication problem that not only enables a drastic reduction of the number of partial products (n/r), but also eliminates the need of pre-computing odd multiples of the multiplicand in higher radix (ß≥8) multiplication. Based on a mathematical proof that any higher radix ß=2^r can be recursively derived from a combination of two or a number of lower radices, a series of generalized radix ß=2^r multipliers are generated by means of primary radices: 2^1 , 2^2, 2^5, and 2^8. A variety of higher-radix (2^3 - 2^{32}) two's complement 64x64 bit serial/parallel multipliers are implemented on Virtex-6 FPGA and characterized in terms of multiply-time, energy consumption per multiply-operation, and area occupation for *r* value varying from 2 to 64. Compared to reference algorithm, savings of 8%, 52%, 63% are respectively obtained in terms of speed, power, and area. In addition, a new low-power and highly-flexible radix 2^r adapted technique for a multi-precision multiplication is presented.**

***Index Terms*— High-Radix Multiplication, Low-Power Multiplication, Multibit Recoding Multiplication, Multi-Precision Multiplication, Partial Product Generator (PPG)**

I. Background and Motivation

In multiplication-intensive applications, as in digital signal processing or process control, multiply-time is a critical factor that limits the whole system performance. When these types of applications are embedded, energy consumption per multiply operation becomes an additional critical issue. Furthermore, in high-precision or large-operand-size applications such as in cryptography, the need for a scalable serial/parallel multiplier is essential as the multiplier size grows quadratically $O(n^2)$ with operand size *n*. Consequently, *high-speed*, *low-power*, and *highly-scalable* architecture are the three major requirements for today's general purpose multiplier [1].

The continuous refinement of the mostly-used design paradigm based on modified Booth algorithm [2] combined to a reduction tree (carry-save-adder array , Dadda[3], HPM[4]) has reached saturation. In [5] and [6] only slight improvements are achieved. Both proposals reduce the partial product number from $n/2+1$ to $n/2$ using different circuit optimization techniques of the critical path.

Theoretically, only the signed multibit recoding multiplication algorithm [7] is capable of a drastic reduction (n/r) of the partial product number, given that $r+1$ is the number of bits of the multiplier that are simultaneously treated ($1\leq r\leq n$). Unfortunately, this algorithm requires the pre-computation of a number of odd multiples of the multiplicand (until $(2^{r-1}-1).X$) that scales linearly with *r*. The large number of odd multiples not only requires a considerable amount of multiplexers to perform the necessary complex recoding into PPG, but dramatically increases the routing density as well. Therefore, a reverse effect occurs that offsets speed and power benefits of the compression factor (n/r). This is the main reason why the multibit recoding algorithm was abandoned. In practice, designs do not exceed r=3 (radix 8).

The current trend [8][9] relies upon advanced arithmetic to determine minimal numeric bases that are representatives of the digits resulting from larger multibit recoding. The objective is to eliminate information redundancy inside $r+1$ bit-length slices for a more compact PPG. This is achievable as long as no or just very few odd multiples are required.

In [8], Seidel et al. have introduced a secondary recoding of digits issued from an initial multibit recoding for $5\leq r\leq 16$. The recoding scheme is based on balanced complete residue system. Though it significantly reduces the number of partial products (n/r for $5\leq r\leq 16$), it requires some odd multiples for $r\geq 8$. While in [9], Dimitrov et al. have proposed a new recoding scheme based on double base number system for $6\leq r\leq 11$. The algorithm is limited to unsigned multiplication and requires a larger number of odd multiples.

Instead of looking for more effective numeric bases, which is a hard mathematical task, our approach consists in exploiting already existing odd-multiple *free* recoding algorithms (2^1 , 2^2, 2^5, and 2^8) to recursively build up generalized odd-multiple *free* radix 2^r recoding schemes.

To achieve such a goal, the multibit recoding multiplication algorithm is revisited [7]. Its design space is extended by the introduction of a new recursive version that enables a hardware-friendly space-time partitioning of the multiplication problem. Depending on *r* value ranging from 2 to *n*, highly-scalable signed multipliers with various levels of parallelism and latencies can be systematically generated with insignificant control-complexity. The new algorithm has also the merit to recursively reduce the number of partial products (n/r) without any limit for the parameter *r* and any need for the odd multiples of the multiplicand. It also allows the combination of different recoding schemes proposed in the literature into the same architecture for better performances of the multiplier. Several higher radix (ß=2^3, 2^{32}) two's complement 64x64 bit serial/parallel multipliers based on combined recoding schemes are implemented on Virtex-6 FPGA and characterized in terms of speed, power, and area occupation for *r* values ranging from 2 to 64. Compared to a new signed version of Dimitrov et al. algorithm [9] and Seidel et al. algorithm [8], outstanding results are obtained with the new multibit recoding scheme for r=8 formed by the combination of Seidel algorithm (r=5), MacSorley algorithm (r=2) [2] and Booth algorithm (r=1) [10]. The respective savings are as follows: 21%, 53%, 105% and 8%, 52%, 63% are obtained in terms of multiply-time, energy consumption per multiply-operation, and total gate count, respectively. In addition, a new low-power and high-throughput radix 2^r adapted technique for multi-precision multiplication is introduced. Contrary to existing techniques [11][12], this new one allows a customized partitioning of the operands in any number of sub-operands and in any sub-operand bit-sizes.

The paper is organized as follows. Section I outlines the main requirement specifications for a generalized radix 2^r multiplication. Section II introduces the new recursive multibit recoding multiplication algorithm. Afterwards, some high-radix (ß=2^3, 2^8) variants of the new algorithm are presented in Section III, while their implementation results are discussed in Section IV. Higher radix (ß=2^8, 2^{32}) algorithms are introduced in Section V. Section VI describes the new low-power technique for multi-precision multiplication. Finally, Section VII provides some concluding remarks and suggestions for future work.

II. The New Recursive Multibit Recoding Multiplication Algorithm

The equation (2.1.2) of the original multibit recoding algorithm presented in [7] (see Appendix) does not offer hardware visibility. Let us rewrite it in a simpler hardware-friendly form, as follows:

$$Y=\sum_{j=0}^{\frac{n}{r}-1}\left(y_{rj-1}+2^0 y_{rj}+2^1 y_{rj+1}+2^2 y_{rj+2}+\cdots+2^{r-2} y_{rj+r-2}-2^{r-1} y_{rj+r-1}\right)2^{rj}=\sum_{j=0}^{\frac{n}{r}-1} Q_j 2^{rj} \qquad (1)$$

Where $y_{-1}=0$ and $r \in \mathrm{N}^*$. For simplicity purposes and without loss of generality, we assume that r is a divider of n.

In this general case, the multiplier Y is split into n/r slices, each of r+1 bit length. Each pair of two contiguous slices has one overlapping bit. In literature, equation (1) is referred to by radix $ß=2^r$, to which corresponds a digit set $D(2^r)$ such as $Q_j \in D(2^r)=\{-2^{r-1},\ldots,0,\ldots,2^{r-1}\}$.

Thus, the signed multiplication between X and Y becomes:

$$X.Y=\sum_{j=0}^{\frac{n}{r}-1} X.Q_j.2^{rj} \qquad (2)$$

Where each partial product can be expressed as follows: $X.Q_j.2^{rj}=(-1)^s.2^e.(m.X)$, with $m \in O_m(2^r)=\{\ 1,3,\ \ldots,\ 2^{r-1}-1\ \}$. $O_m(2^r)$ represents the required set of odd multiples of the multiplicand ($m.X$) for radix 2^r . Hence, the partial product generation process consists first in selecting one odd multiple ($m.X$) among the whole set of pre-computed odd multiples, which is then submitted to a hardwired shift of e positions, and finally conditionally complemented $(-1)^s$ depending on the bit sign s of Q_j term. Table I provides a picture on how the number of odd multiples grows when the radix becomes higher. While lower $m.X$ can be obtained using just one addition (3X=2X+1X), the calculation of higher ones may require a number of computation steps (11X= 8X+2X+1X).

TABLE I
Main Features of the Multibit Recoding Multiplication Algorith

Radix	Nbr. of Partial Products	Odd Multiples ($m.X$)
2^1	n	1X
2^2	$n/2$	1X
2^3	$n/3$	1X, 3X
2^4	$n/4$	1X, 3X, 5X, 7X
2^5	$n/5$	1X, 3X, 5X, 7X, 9X, 11X, 13X, 15X

In radix 2^r, the multiplier Y is divided into n/r slices, each of r+1 bit length. Each pair of two contiguous slices has one overlapping bit.

To bypass the hard problem of odd multiples, let us announce the two following theorems accompanied with their respective proofs:

Theorem 1. *Any digit $Q_j \in D(2^r)$ can be represented in a combination of digits $P_{ji} \in D(2^s)$, such as s is a divider of r.*

Proof. Equation (1) is recursively applied on Q_j term of equation (1). Thus, equation (1) becomes:

$$\begin{aligned}Y=\sum_{j=0}^{\frac{n}{r}-1}\Big[&\big(y_{rj-1}+2^0 y_{rj}+2^1.y_{rj+1}+2^2 y_{rj+2}+\cdots\\&+2^{s-2} y_{rj+s-2}-2^{s-1} y_{rj+s-1}\big)\ 2^0+\\&\big(y_{rj+s-1}+2^0 y_{rj+s}+2^1.y_{rj+s+1}+2^2 y_{rj+s+2}+\cdots\\&+2^{s-2} y_{rj+2s-2}-2^{s-1} y_{rj+2s-1}\big)\ 2^s+\\&\vdots\\&+\big(y_{rj+r-1-2s}+2^0 y_{rj+r-2s}+2^1.y_{rj+r-2s+1}+2^2 y_{rj+r-2s+2}+\cdots\\&+2^{s-2} y_{rj+r-s-2}-2^{s-1} y_{rj+r-1-s}\big)\ 2^{s\left(\frac{r}{s}-2\right)}+\\&\big(y_{rj+r-1-s}+2^0 y_{rj+r-s}+2^1.y_{rj+r+s+1}+2^2 y_{rj+r+s+2}+\cdots\\&+2^{s-2} y_{rj+r-2}-2^{s-1} y_{rj+r-1}\big)\ 2^{s\left(\frac{r}{s}-1\right)}\Big]2^{rj}\end{aligned}$$

$$=\sum_{j=0}^{\frac{n}{r}-1}\left[\sum_{i=0}^{\frac{r}{s}-1}\left(y_{rj-1+si}+2^0 y_{rj+si}+2^1 y_{rj+1+si}+2^2 y_{rj+2+si}+\ldots+2^{s-2} y_{rj+s-2+si}-2^{s-1} y_{rj+s-1+si}\right)2^{si}\right]2^{rj}$$

$$=\sum_{j=0}^{\frac{n}{r}-1}\left[\sum_{i=0}^{\frac{r}{s}-1} P_{ji}\ 2^{si}\right]2^{rj} \qquad (3)$$

Where $Q_{ji} \in D(2^s)=\{-2^{s-1},\ldots,0,\ldots,2^{s-1}\}$ with $m \in O_m(2^s)=\{\ 1,3,\ \ldots,\ 2^{s-1}-1\ \}$ and

$$X.Y=\sum_{j=0}^{\frac{n}{r}-1}\left[\sum_{i=0}^{\frac{r}{s}-1} X.P_{ji}\ 2^{si}\right]2^{rj} \qquad (4)$$

Theorem 2. *Any digit $Q_j \in D(2^r)$ can be represented in a combination of digits $P_{ji}+T_{jk}$ such as $P_{ji} \in D(2^s)$ and $T_{jk} \in D(2^t)$ with s+t a divider of r , and t < s.*

Proof. Equation (1) can also be rewritten as follows:

$$\begin{aligned}Y=\sum_{j=0}^{\frac{n}{r}-1}\Bigg[\sum_{i=0}^{\frac{r}{s+t}-1}\Big[&\big(y_{rj-1+(s+t)i}+2^0 y_{rj+(s+t)i}+2^1.y_{rj+1+(s+t)i}+\cdots\\&+2^{s-2} y_{rj+s-2+(s+t)i}-2^{s-1} y_{rj+s-1+(s+t)i}\big)\ +\\&\big(y_{rj+s-1+(s+t)i}+2^0 y_{rj+s+(s+t)i}+2^1.y_{rj+1+s+(s+t)i}+\cdots\\&+2^{t-2} y_{rj+r-2+(s+t)i}-2^{t-1} y_{rj+r-1+(s+t)i}\big)\ 2^s\Big]2^{(s+t)i}\Bigg]\ 2^{rj}\end{aligned}$$

$$= \sum_{j}^{\frac{n}{r}-1} \left[\sum_{i=0}^{\frac{r}{s+t}-1} \left[P_{ji} + T_{ji}\, 2^{s} \right] 2^{(s+t)i} \right] 2^{rj} \qquad (5)$$

Where $P_{ji} \in D(2^s) = \{-2^{s-1}, \dots, 0, \dots, 2^{s-1}\}$ with $m \in O_m(2^s) = \{1, 3, \dots, 2^{s-1}-1\}$ and $T_{ji} \in D(2^t) = \{-2^{t-1}, \dots, 0, \dots, 2^{t-1}\}$ with $m \in O_m(2^t) = \{1, 3, \dots, 2^{t-1}-1\}$ and

$$X.Y = \sum_{j}^{\frac{n}{r}-1} \left[\sum_{i=0}^{\frac{r}{s+t}-1} \left[X.P_{ji} + X.T_{ji}\, 2^{s} \right] 2^{(s+t)i} \right] 2^{rj} \qquad (6)$$

As a result of theorems (1) and (2), much less odd multiples are needed in partial products of equations (4) and (6) than in equation (2), but at the expense of a number of additions. The advantage by far outweighs the cost, as practically shown in the next section. The translation of equation (4) into architecture is depicted by Fig. 1, where each PPG$_j$ is built up using identical PPG$_{ji}$. This is not the case for equation (6) which requires two different PPG$_{ji}$. Theorem (1) and (2) can be merged to produce PPG$_j$ made of a number of different PPG$_{ji}$.

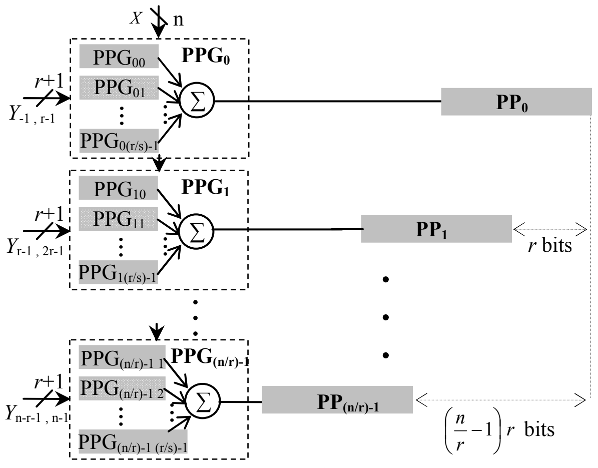


Fig. 1. Generalized n×n bit radix 2^r parallel multiplier based on sub-radix 2^s. Space partitioning according to *r* and s values

III. Some Variants of the New Recursive Multibit Recoding Multiplication Algorithm

Theorems (1) and (2) permit to build up any higher radix multiplication algorithm based on lower radices. But the objective is to generate higher radix multiplication without odd multiples. To achieve such a goal, a number of odd-multiple free low-radix algorithms are used, such as Booth algorithm [10] (radix 2^1), modified Booth algorithm [2] (radix 2^2), Seidel et al. algorithms [8][13] (radix 2^5 and radix 2^8).

Booth and modified Booth recoding are respectively derived form equation (3) for (*r*,*s*)=(1,1) and (*r*,*s*)=(2,2). They are respectively summarized as follows:

$$Y = \sum_{j=0}^{n-1} \left(y_{j-1} - y_j \right) 2^j = \sum_{j=0}^{n-1} Q_j\, 2^j \qquad (7)$$

With $D(2) = \{-1, 0, 1\}$ and $O_m(2) = \{1\}$

$$Y = \sum_{j=0}^{(n/2)-1} \left(y_{2j-1} + y_{2j} - 2y_{2j+1} \right) 2^{2j} = \sum_{j=0}^{(n/2)-1} Q_j\, 2^{2j} \qquad (8)$$

With $D(2^2) = \{-2, -1, 0, 1, 2\}$ and $O_m(2^2) = \{1\}$

Higher radices are obtained as follows.

A. Radix 2^3 recoding

Radix 2^3 recoding based on equation (1) gives:

$$Y = \sum_{j=0}^{(n/3)-1} \left(y_{3j-1} + y_{3j} + 2y_{3j+1} - 2^2 y_{3j+2} \right) 2^{3j} = \sum_{j=0}^{(n/3)-1} Q_j\, 2^{3j}$$

With $D(2^3) = \{-4, \dots, 0, \dots, 4\}$ and $O_m(2^3) = \{1, 3\}$

While radix 2^3 recoding based on equation (5) delivers:

$$Y = \sum_{j=0}^{(n/3)-1} \left[\left(y_{3j-1} + 2^0 y_{3j} - 2^1 y_{3j+1} \right) + \left(y_{3j+1} - y_{3j+2} \right) 2^2 \right] 2^{3j}$$

$$= \sum_{j=0}^{(n/3)-1} \left[P_j + T_j\, 2^2 \right] 2^{3j} \qquad (9)$$

In fact, equation (9) is a combination of Booth ($O_m(2^1) = \{1\}$) and modified Booth algorithms ($O_m(2^2) = \{1\}$). Hence, for equation (9), $O_m(2^3) = \{1\}$.

Furthermore, equation (9) is recursively used to generate any radix 2^r recoding with $O_m(2^r) = \{1\}$ based on radix 2^3, as follows:

$$Y = \sum_{j}^{\frac{n}{r}-1} \left[\sum_{i=0}^{\frac{r}{3}-1} \left[\left(y_{3j-1+3i} + 2^0 y_{3j+3i} - 2^1 y_{3j+1+3i} \right) + \right. \right.$$

$$\left. \left. + \left(y_{3j+1+3i} - y_{3j+2+3i} \right) 2^2 \right] 2^{3i} \right] 2^{rj}$$

$$= \sum_{j}^{\frac{n}{r}-1} \left[\sum_{i=0}^{\frac{r}{3}-1} \left[P_{ji} + T_{ji}\, 2^2 \right] 2^{3i} \right] 2^{3j} \qquad (ß2^3)$$

This equation is referred to by ß2^3 for later comparison with other general radix algorithms based on lower radices. Its corresponding architecture is illustrated by Fig. 2.

B. Radix 2^4 recoding

For *r*=4, equation (1) needs an $O_m(2^4) = \{1, 3, 5, 7\}$, while equation (3) with (*r*,*s*)=(4,2) requires an $O_m(2^4) = \{1\}$. Three odd multiples are eliminated at the expense of one addition. The general radix (2^r) recoding with $O_m(2^r) = \{1\}$ based on radix 2^2 is: $Y = \sum_{j=0}^{\frac{n}{r}-1} \left[\sum_{i=0}^{\frac{r}{2}-1} Q_{ji}\, 2^{2i} \right] 2^{rj}$ $\quad (ß2^2)$

C. Radix 2^5 recoding

For *r*=5, eq. (1) needs an $O_m(2^5) = \{1, 3, 5, 7, 9, 11, 13, 15\}$. In this case we do recourse to Seidel et al. algorithm [8][13]:

$$Y = \sum_{j=0}^{(n/5)-1} \left[7.Q_j + P_j \right] 2^{5j} \qquad (10)$$

With $Q_j \in \{-2, -1, 0, 1, 2\}$; $P_j \in \{-4, -2, -1, 0, 1, 2, 4\}$ and $O_m(2^5) = \{1\}$

Equation (10) is integrated into equation (3), which gives the following general recursive form with $O_m(2^r) = \{1\}$:

$$Y = \sum_{j=0}^{\frac{n}{r}-1} \left[\sum_{i=0}^{\frac{r}{5}-1} \left[7.Q_{ji} + P_{ji} \right] 2^{5i} \right] .2^{rj} \qquad (ß2^5)$$

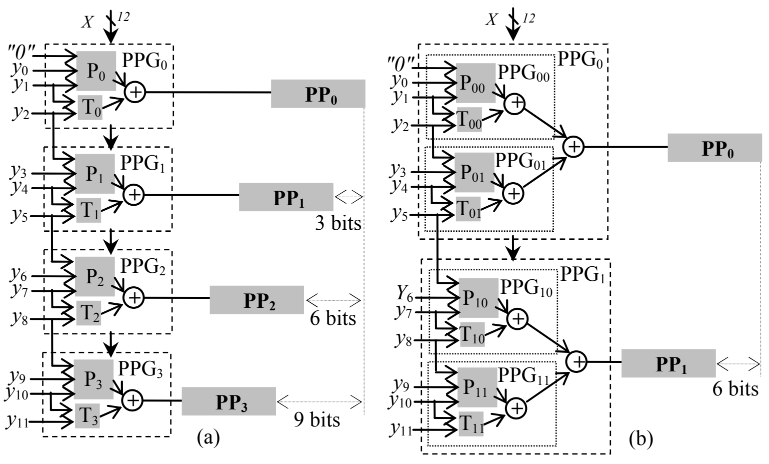


Fig. 2. Two's complement 12×12 bit parallel multiplier based on equation (ß2^3). Space partitioning according to: r=3 (a) and r=6 (b)

As application, radix 2^{10}= 1024 with $O_m\left(2^{10}\right)=\{1\}$ and just five additional adders is obtained with:

$$Y=\sum_{j=0}^{(n/10)-1}\left[\left(7.Q_{j0}+P_{j0}\right)+\left(7.Q_{j1}+P_{j1}\right)2^5\right].2^{rj}$$

D. *Radix 2^8 recoding*

Seidel et al. [13] recoding for radix 2^8 with $O_m\left(2^8\right)=\{1\}$ is: $Y=\sum_{j=0}^{(n/8)-1}\left[11^2.Q_j+11.P_j+T_j\right]2^{8j}$ (11)

With $Q_j\in\{-2,-1,0,1,2\}$ *and*

$P_j\ ,T_j\in\{-16,-8,-4,-2,-1,0,1,2,4,8,16\}$

Equation (11) is incorporated into equation (3) to obtain the general recursive form with $O_m\left(2^r\right)=\{1\}$. It gives:

$$Y=\sum_{j=0}^{\frac{n}{r}-1}\left[\sum_{i=0}^{\frac{r}{8}-1}\left[11^2.Q_{ji}+11.P_{ji}+T_{ji}\right]2^{8i}\right].2^{rj}\qquad (ß2^8)$$

For performance comparison, we developed a new signed radix 2^8 recoding with $O_m\left(2^8\right)=\{1,3,5,7\}$ based on unsigned radix 2^7 recoding with $O_m\left(2^7\right)=\{1,3,5,7\}$ proposed by Dimitrov et al. in [9]. The new recoding is:

$$Y=\sum_{j=0}^{(n/8)-1}\left(2^k.Q_j+(-1)^e\,2^h.P_j\right)2^{8i}\qquad (12)$$

With $Q_j,P_j\in\{1,3,5,7\};k,h\in\{0,1,2,3,4,5,6,7\}$ *and* $e\in\{0,1\}$

Likewise, equation (12) is integrated into equation (3). The general recursive form with $O_m\left(2^r\right)=\{1,3,5,7\}$ becomes:

$$Y=\sum_{j=0}^{\frac{n}{r}-1}\left[\sum_{i=0}^{\frac{r}{8}-1}\left(2^k.Q_{ji}+(-1)^e\,2^h.P_{ji}\right)2^{8i}\right].2^{rj}\qquad (ß'2^8)$$

IV. Discussion of the Implementation Results

In the preceding section, we introduced five generalized multibit space-time partitioning schemes, which are: ß2^2, ß2^3, ß2^5, ß2^8, and ß'2^8. They all require $O_m\left(2^r\right)=\{1\}$ except ß'2^8 that needs $O_m\left(2^r\right)=\{1,3,5,7\}$.

In this paper, only the serial/parallel form is explored (Fig. 3), targeting applications where the serialization of multiplication is mandatory. This is the case for instance in embedded digital PID (Peripheral Integral Derivative) controller where five multiplication cores are required [14], or for high-precision or very large operand size applications (cryptography) where a fully-parallel n×n bit implementation is excluded.

In signed serial/parallel multiplication, r-bit slices of the multiplier are processed each clock cycle, which induces a theoretical multiply time of n/r for a double precision product (2n bits). The special cases where r=n and r=r_{min} correspond to fully-parallel and fully-sequential multiplier, respectively. In between (r=2r_{min} , n/2), partially-parallel multipliers are obtained. In fact, the lower limit of r depends on the recoding scheme used (ex: for ß2^5, r_{min}=5). Reader is referred to [8], [13], and [9] for recoding tables used respectively in ß2^5, ß2^8, and ß'2^8.

Before comparison, all recoding schemes proposed in this paper underwent several steps of verification. First all equations were validated with a random C-program. Then, they were implemented at RTL level in Verilog-2001 (IEEE 1364) as technology-independent reusable IP-cores [1], using *exactly* the same optimized coding style for an equitable comparison. They are compile-time reconfigurable according to n and r. All RTL codes went through a severe cycle-accurate functional verification procedure using Modelsim SE-6.3f logic simulator. They were first challenged against a set of special and severe test cases (visual simulation), and then submitted to a random test for a very large number of vectors. After a successful functional verification, physical tests were performed. They were integrated into an FPGA evaluation board for an ultimate validation.

Afterwards, all equations were synthesized and mapped to the same Virtex-6 FPGA circuit (XC6VSX475t-2FF1156) using Xilinx ISE 13.2 release version [15]. Two's complement 64x64 bit radix 2^r serial/parallel multipliers with r varying from r_{min} to 64 were characterized in terms of area occupation (number of occupied Virtex-6 slices), maximum multiply time, and maximum energy consumption per multiply operation. The results are depicted in Fig. 4, 5 and 6, respectively.

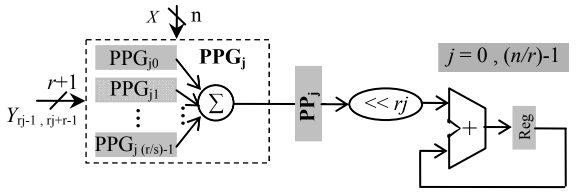


Fig. 3. Generalized radix 2^r serial/parallel multiplier based on sub-radix 2^s . Space-time partitioning according to r and s values

A. *Area Occupation*

Three basic components are necessary for the implementation of the proposed multipliers: a) multiplexers to decode the digit terms Q_{ji} , $P_{ji\ ,\ \ldots\ ;}$ b) shifters for partial product generation; c) and adders for partial product summation. Whereas the exact number of adders can be known in advance, we need to develop heuristics for the two others. Multiplexer complexity depends on: a) the lower radix 2^s used to build up the higher radix 2^r; b) the number (i) of "case" statements used to decode the digit terms; c) the number of entries (e_i) in each "case" statement; d) the number (d_i) of digit terms; e) and on the number of odd multiples ($|O_{mi}|$) used to calculate the digit terms. Hence, we can announce that: $Mux1=\frac{r}{s}.\sum_i\left(2^{e_i}.d_i.|O_{mi}|\right)$. For ß'2^8, this gives: s=8, i=1, e_1=9, d_1=2, and $|O_{m1}|$=4. Thus, *Mux1* = 512 *r*.

The synthesis of the "shift" statement infers multiplexers whose complexity depends on the number of different shift operations (b_i) for all odd-multiples involved in the calculation of each digit term (i). Thus, we can write: $Mux2=\frac{r}{s}.\sum_i\left(b_i.|O_{mi}|\right)$. For ß'2^8, this gives: s = 8, i = 2, b_1 = b_2 = 8, and $|O_{m1}|$ = $|O_{m2}|$ = 4. Thus, *Mux2* = 8*r*. Hence, the total multiplexer complexity becomes: *Mux* = *Mux1*+*Mux2*.

The total number of adders comprises $(r/s)-1$ adders to sum up the r/s partial products, plus one adder to accumulate the n/r intermediate summations (Fig. 3), plus the necessary adders included in the r/s PPGs. This latter depends on the recoding scheme used. For example, in ß2^8 the term $11^2Q_{ji}+11P_{ji}+T_{ji}$ is calculated as follows: $\left(2^7Q_{ji}-2^3Q_{ji}+Q_{ji}\right)+\left(2^3P_{ji}+2^2P_{ji}-P_{ji}\right)+T_{ji}$, which requires 6 adders. Hence, the total number of adders required by ß2^8 is: $Add=(r/8)+6(r/8)=(7/8)r$. Table II provides the area occupation for each recoding algorithm. To determine which factor, *Mux* or *Add*, exerts more influence on the area occupation, let us compare their respective ratios for ß2^8 and ß'2^8: $Mux(\beta'2^8)/Mux(\beta2^8)=2.7$ and $Add(\beta2^8)/Add(\beta'2^8)=3.5$

Significant conclusion: the area occupation is dominated by the *Mux* factor, and becomes larger (Fig. 4) as *Mux* number becomes higher (Table II). This correlation is advantageously used to minimize the area occupation and power consumption as will be shown in the next section.

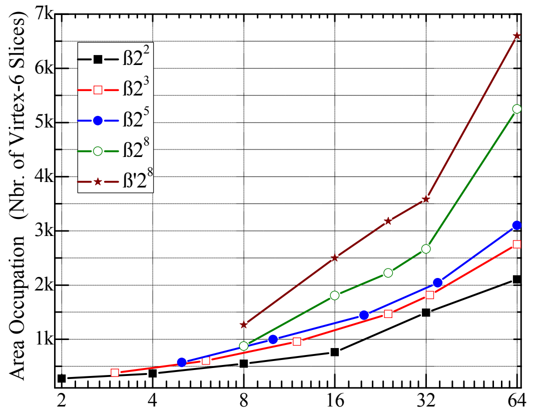


Fig. 4. Area occupation versus *r*

TABLE II
THEORETICAL ESTIMATION OF AREA OCCUPATION AND DELAY

Recoding Algorithm	Area Occupation		Delay (levels)				
				Linear		Logarithmic	
	Mux	*Add*	*PPG delay*	*PPG Adders*	*Reduction Tree*	*PPG Adders*	*Reduction Tree*
ß2^2	$5r$	$r/2$	d_2	0	$r/2$	0	$\log_2(r/2)+1$
ß2^3	$5r$	$(2/3)r$	d_2	1	$r/3$	1	$\log_2(r/3)+1$
ß2^5	$27r$	$(3/5)r$	d_5	2	$r/5$	2	$\log_2(r/5)+1$
ß2^8	$194r$	$(7/8)r$	d_8	6	$r/8$	3	$\log_2(r/8)+1$
ß'2^8	$520r$	$r/4$	d'_8	1	$r/8$	1	$\log_2(r/8)+1$

d_i is the critical delay through PPG. It depends on *Mux* factor ($d_2 < d_5 < d_8 < d'_8$)

B. Energy consumption

While energy consumption is function of the switched capacitance, Fig. 4 and 5 show a direct correlation between area occupation and energy consumption. Making *Mux* indicator lower, will result in a less energy-consumer recoding algorithm.

C. Delay

The delay (T) along the critical path is the summation of PPG delay and reduction tree delay. While the former is constant, the latter depends on the topology used: either linear or logarithmic. The number of levels for each case is given in Table II and the performance of each algorithm is depicted in Fig. 6. The total multiply time is equal to (n/r)T. Note that all results presented in this paper are based on linear implementation of the reduction tree.

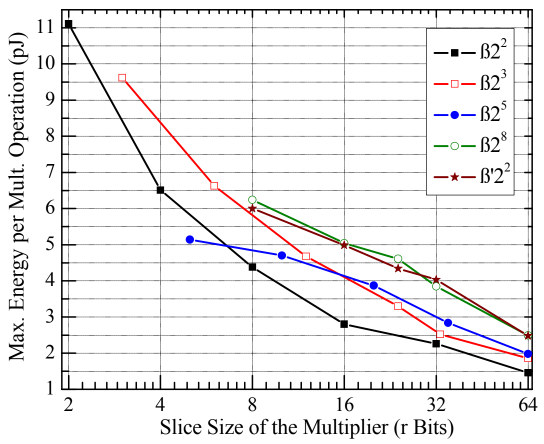


Fig. 5. Max. energy consumption per mult. operation versus *r*

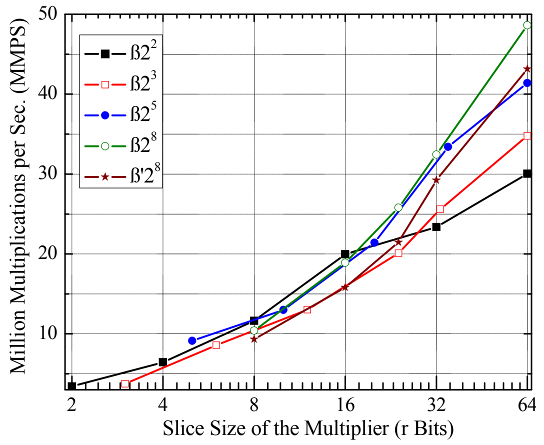


Fig. 6. Max. multiply operations versus *r*

Based on theory and implementation results, it is set clear that ß2^2 algorithm is the best in terms of area and energy consumption. As for speed, ß2^2 is the fastest until r=16. Beyond this value, it is surpassed by ß2^8. ß2^2 algorithm served to design a scalable 16-bit set-point PID controller employing five multiplication cores. The implementation results outperformed the published ones at all levels [14].

V. HIGHER RADIX MULTIBIT RECODING MULTIPLICATION ALGORITHMS

Further performance requires higher r values ($r \geq 8$) necessarily. Guided by *Mux* and *Add* indicators, the objective is to generate a recoding scheme that outperforms ß2^2 in area and power, and ß2^8 in speed.

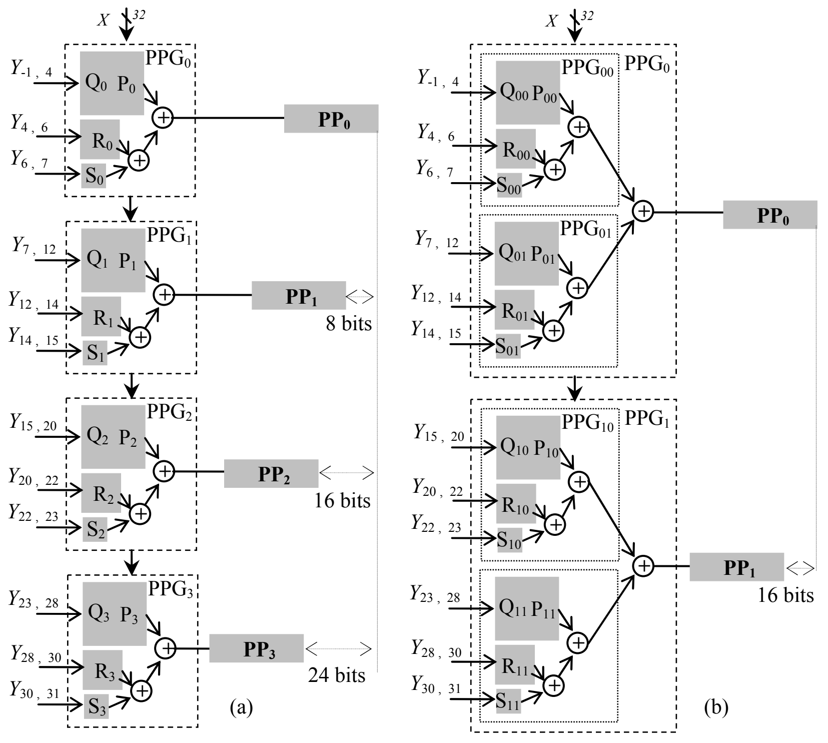


Fig. 7. Two's complement 32×32 bit multiplier based on equation (ß"2^8). Space partitioning according to: r=8 (a) and r=16 (b)

A. *Radix 2^8 recoding*

Based on theorem (2), ß2^3 and ß2^5 are merged to build up a new (ß"2^8) radix 2^8 recoding algorithm (Table III and Fig. 7). ß"2^8 exhibits (Table IV) *Mux* and *Add* values of 19r and $r/4$, respectively. It outperforms ß2^8 at all aspects (Fig. 8, 9, 10). Result summary with regard to Dimitrov and Seidel algorithms is given in Table V.

B. *Radix 2^{13} recoding*

As ß2^8 and ß2^5 show good results for speed and power respectively, they have been merged (ß2^{13}) for a better compromise. However, the *Mux* saving (130r) is not important enough compared to *Mux* value (192r) of ß2^8. This explains the closeness of the results between ß2^{13} and ß2^8.

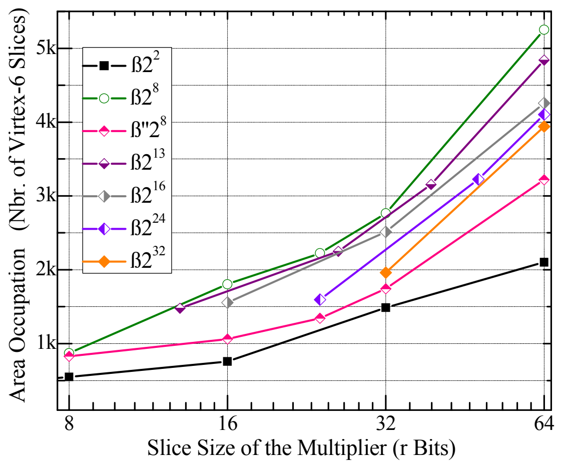


Fig. 8. Area occupation versus r

C. *Radix 2^{16} recoding*

To achieve a significant *Mux* saving, ß2^8 is combined with ß2^2 based on theorem (1) and (2) simultaneously. ß2^{16} exhibits a *Mux* value of 100r, which is almost the half required by ß2^8. Better results are obtained in terms of area and energy. The fact that ß2^{16} is little bit slower than ß2^8 is due to the higher PPG adder number required (10). For r greater than 64, ß2^{16} will surpasses ß2^8 since the total number of adder levels will be lower. Higher radices provide more speed.

D. *Radix 2^{24} recoding*

To push lower the energy consumption while increasing the speed, lower *Mux* values with higher radices are required. This can be achieved using the mixture of: ß2^8 , ß2^5 , ß2^3 , and ß2^2 . In this case, for ß2^{24} *Mux* = 74r, which yields to more interesting results.

E. *Radix 2^{32} recoding*

Even more attractive results (*Mux*=60r) are obtained with ß2^{32} composed of ß2^8 , ß2^5 , ß2^3 , and ß2^2 (Fig. 11). At this level, some useful conclusions can be drawn depending on the topology of the reduction tree used, either linear or logarithmic (Table VI). In the case of a linear tree, ß2^2 is the most area and energy efficient algorithm for any value of r. For r ranging from 8 to 64, ß"2^8 is the fastest algorithm, but it will be outperformed by ß2^{32} for r values greater than 64. In the case of logarithmic reduction tree, ß2^2 is by far the best at all aspects since it always requires the lowest number of adder levels $(d_2+\log_2(r/2)+1)$ whatever r value (Table VI).

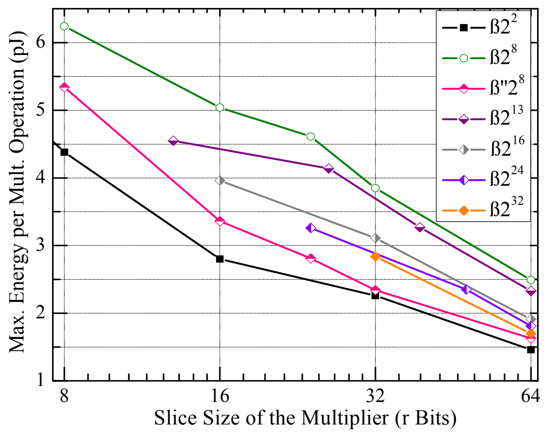


Fig. 9. Max. energy consumption per mult. operation versus r

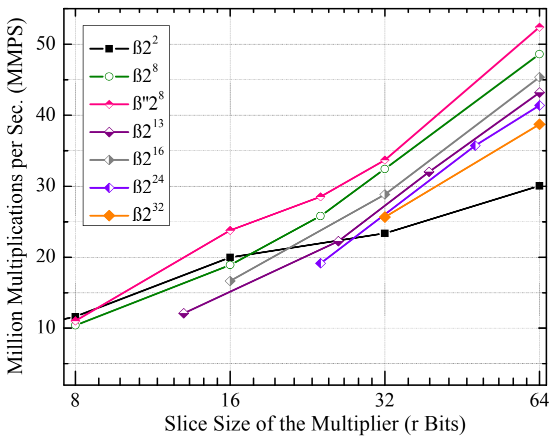


Fig. 10. Max. multiply operations versus r

TABLE III
SUMMARY OF OUR NEW RADIX-2^r MULTIBIT RECODING ALGORITHMS

Recoding Algorithm	Recoding Equation and Main Features
$\beta 2^r$	$Y=\sum_{j=0}^{\frac{N}{r}-1} Q_j . 2^{rj}$; BR: 2^r ; OM: $\{1,3,\ldots,2^{r-1}-1\}$; DV: $Q_j \in \{-2^{r-1},\ldots,0,\ldots,2^{r-1}\}$;
$\beta 2^2$	$Y=\sum_{j=0}^{\frac{N}{r}-1}\left[\sum_{i=0}^{\frac{r}{2}-1} Q_{ji}\, 2^{2i}\right].2^{rj}$; BR: 2^2 ; OM: $\{1\}$; DV: $Q_{ji} \in \{-2,-1,0,1,2\}$;
$\beta 2^3$	$Y=\sum_{j}^{\frac{N}{r}-1}\left[\sum_{i=0}^{\frac{r}{3}-1}\left(Q_{ji}+P_{ji}\, 2^2\right).2^{3i}\right].2^{rj}$; BR: $2^1, 2^2$; OM: $\{1\}$; DV: $Q_{ji} \in \{-2,-1,0,1,2\}$, $P_{ji} \in \{-1,0,1\}$
$\beta 2^5$	$Y=\sum_{j=0}^{\frac{N}{r}-1}\left[\sum_{i=0}^{\frac{r}{5}-1}\left(7.Q_{ji}+P_{ji}\right)2^{5i}\right].2^{rj}$; BR: 2^5 ; OM: $\{1\}$; DV: $Q_{ji} \in \{-2,-1,0,1,2\}$, $P_{ji} \in \{-4,-2,-1,0,1,2,4\}$
$\beta 2^8$	$Y=\sum_{j=0}^{\frac{N}{r}-1}\left[\sum_{i=0}^{\frac{r}{8}-1}\left(11^2.Q_{ji}+11.P_{ji}+T_{ji}\right)2^{8i}\right].2^{rj}$; BR: 2^8 ; OM: $\{1\}$; DV: $Q_{ji} \in \{-2,-1,0,1,2\}$, $P_{ji}\ , T_{ji} \in \{-16,-8,-4,-2,-1,0,1,2,4,8,16\}$;
$\beta' 2^8$	$Y=\sum_{j=0}^{\frac{N}{r}-1}\left[\sum_{i=0}^{\frac{r}{8}-1}\left(2^k\ .\ Q_{ji}+(-1)^e\, 2^h.P_{ji}\right)2^{8i}\right].2^{rj}$ BR: 2^8 ; OM: $\{1,3,5,7\}$; DV: $Q_{ji}\ , P_{ji} \in \{1,3,5,7\}$ $k\ , h \in \{0,1,2,3,4,5,6,7\}$; $e \in \{0,1\}$
$\beta'' 2^8$	$Y=\sum_{j=0}^{\frac{N}{r}-1}\left[\sum_{i=0}^{\frac{r}{8}-1}\left[\left(7.Q_{ji}+P_{ji}\right)+\left(R_{ji}+S_{ji}\, 2^2\right).2^5\right].2^{8i}\right].2^{rj}$ BR: $2^1, 2^2, 2^5$; OM: $\{1\}$; DV: $Q_{ji} \in \{-2,-1,0,1,2\}$, $P_{ji} \in \{-4,-2,-1,0,1,2,4\}$, $R_{ji} \in \{-2,-1,0,1,2\}$, $S_{ji} \in \{-1,0,1\}$
$\beta 2^{13}$	$Y=\sum_{j=0}^{\frac{N}{r}-1}\left[\sum_{i=0}^{\frac{r}{13}-1}\left[\left(11^2.Q_{ji}+11.P_{ji}+T_{ji}\right)+\left(7.R_{ji}+S_{ji}\right).2^8\right].2^{13i}\right].2^{rj}$ BR: $2^5, 2^8$; OM: $\{1\}$; DV: $Q_{ji} \in \{-2,-1,0,1,2\}$, $P_{ji}\ , T_{ji} \in \{-16,-8,-4,-2,-1,0,1,2,4,8,16\}$, $R_{ji} \in \{-2,-1,0,1,2\}$, $S_{ji} \in \{-4,-2,-1,0,1,2,4\}$
$\beta 2^{16}$	$Y=\sum_{j=0}^{\frac{N}{r}-1}\left[\sum_{i=0}^{\frac{r}{16}-1}\left[\left(11^2.Q_{ji}+11.P_{ji}+T_{ji}\right)+\left(\sum_{k=0}^{3} M_{kji}.2^{2k}\right).2^8\right].2^{16i}\right].2^{rj}$ BR: $2^2, 2^8$; OM: $\{1\}$; DV: $Q_{ji} \in \{-2,-1,0,1,2\}$, $P_{ji}\ , T_{ji} \in \{-16,-8,-4,-2,-1,0,1,2,4,8,16\}$, $M_{kji} \in \{-2,-1,0,1,2\}$
$\beta 2^{24}$	$Y=\sum_{j=0}^{\frac{N}{r}-1}\left[\sum_{i=0}^{\frac{r}{24}-1}\left[\left(11^2.Q_{ji}+11.P_{ji}+T_{ji}\right)+\left(\sum_{k=0}^{3} M_{kji}.2^{2k}\right).2^8+\left(7.R_{ji}+S_{ji}\right).2^{16}+\left(U_{ji}+V_{ji}\, 2^2\right).2^{21}\right].2^{24i}\right].2^{rj}$ BR: $2^1, 2^2, 2^5, 2^8$; OM: $\{1\}$; DV: $Q_{ji} \in \{-2,-1,0,1,2\}$, $P_{ji}\ , T_{ji} \in \{-16,-8,-4,-2,-1,0,1,2,4,8,16\}$, $M_{kji} \in \{-2,-1,0,1,2\}$, $R_{ji} \in \{-2,-1,0,1,2\}$, $S_{ji} \in \{-4,-2,-1,0,1,2,4\}$, $U_{ji} \in \{-2,-1,0,1,2\}$, $V_{ji} \in \{-1,0,1\}$
$\beta 2^{32}$	$Y=\sum_{j=0}^{\frac{N}{r}-1}\left[\sum_{i=0}^{\frac{r}{32}-1}\left[\left(11^2.Q_{ji}+11.P_{ji}+T_{ji}\right)+\left(\sum_{k=0}^{3} M_{kji}.2^{2k}\right).2^8+\sum_{k=0}^{1}\left(\left(7.R_{kji}+S_{kji}\right).2^{16+8k}+\left(U_{kji}+V_{kji}\, 2^2\right).2^{21+8k}\right)\right].2^{32i}\right].2^{rj}$ BR: $2^1, 2^2, 2^5, 2^8$; OM: $\{1\}$; DV: $Q_{ji} \in \{-2,-1,0,1,2\}$, $P_{ji}\ , T_{ji} \in \{-16,-8,-4,-2,-1,0,1,2,4,8,16\}$, $M_{kji} \in \{-2,-1,0,1,2\}$, $R_{kji} \in \{-2,-1,0,1,2\}$, $S_{kji} \in \{-4,-2,-1,0,1,2,4\}$, $U_{kji} \in \{-2,-1,0,1,2\}$, $V_{kji} \in \{-1,0,1\}$

BR: Based on Radix ; DV: Digit Variations ; OM: Odd-Multiples

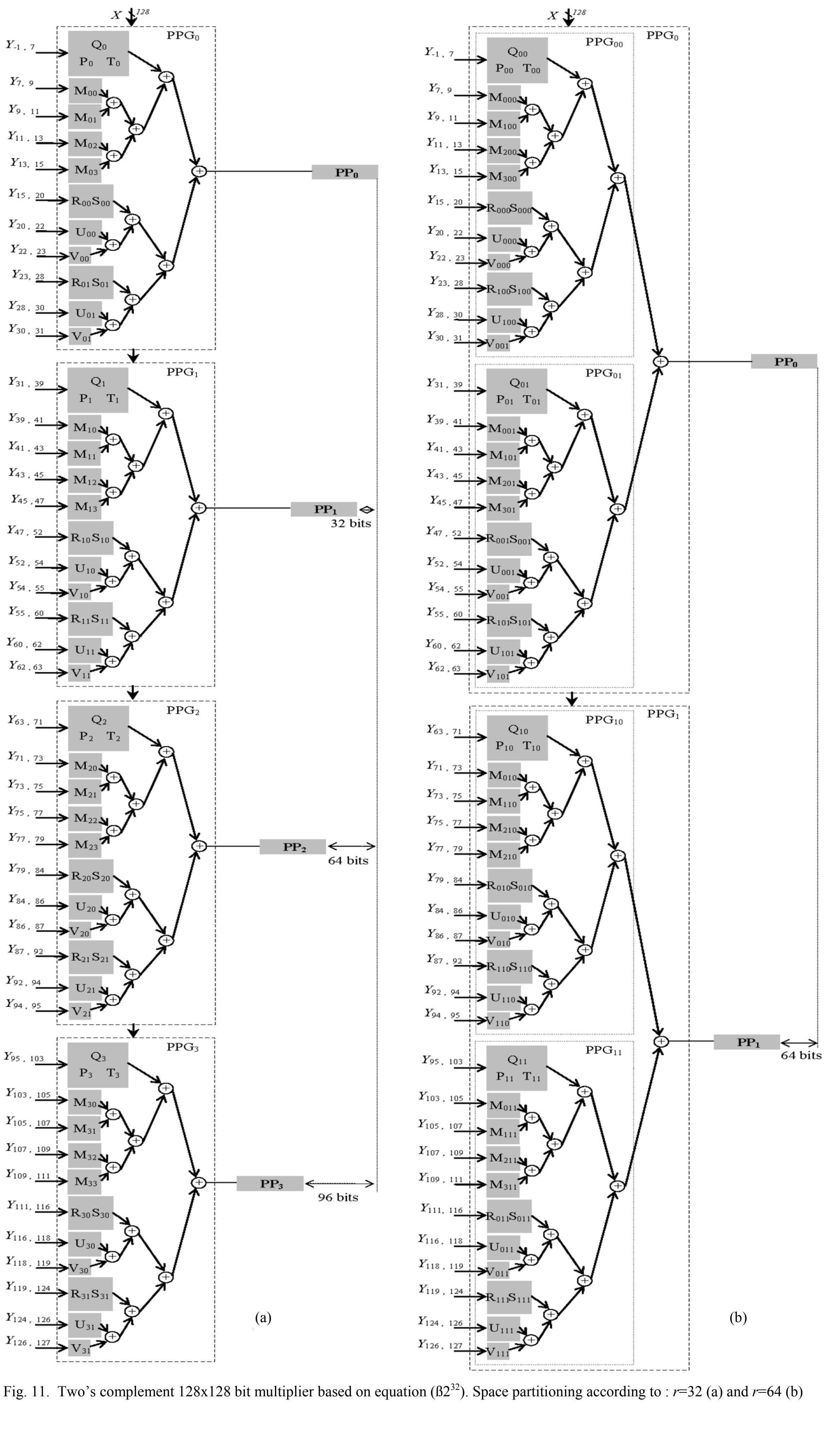


Fig. 11. Two's complement 128x128 bit multiplier based on equation (ß2^{32}). Space partitioning according to : r=32 (a) and r=64 (b)

TABLE IV
THEORETICAL ESTIMATION OF AREA OCCUPATION AND DELAY

Recoding Algorithm	Area Occupation		Delay (levels)				
				Linear		Logarithmic	
	Mux	*Add*	*PPG delay*	*PPG Adders*	*Reduction Tree*	*PPG Adders*	*Reduction Tree*
ß"2^{8}	$19\,r$	$r/4$	d_5	4	$r/8$	3	$\log_2(r/8)+1$
ß2^{13}	$130\,r$	$(8/13)r$	d_8	9	$r/13$	4	$\log_2(r/13)+1$
ß2^{16}	$100\,r$	$(11/16)r$	d_8	10	$r/16$	4	$\log_2(r/16)+1$
ß2^{24}	$74\,r$	$(15/24)r$	d_8	15	$r/24$	4	$\log_2(r/24)+1$
ß2^{32}	$60\,r$	$(21/32)r$	d_8	20	$r/32$	5	$\log_2(r/32)+1$

d_i is the critical delay through PPG. It depends on *Mux* factor ($d_5 < d_8$)

Based on higher radix recoding algorithms proposed so far (ß2^{8}, ß2^{16}, ß2^{24}, and ß2^{32}) as well as on *Mux* and *Add* indicators, the generation process of advanced higher radix algorithms (ß2^{64}, ß2^{128}, ß2^{256},…) with $O_m = \{1\}$can be recursively pursued farther for very large-operand-size applications (n >>). The number of adder levels required by a ß2^{x} algorithm will be: $a + \log_2(r/x) + 1$, where a is a constant depending on the level number of PPG adders. Thus, ß2^{x} will outperform ß2^{2} for $x \geq 4e^{a}$.

TABLE V
RESULT COMPARISON FOR $r = 64$

Implem-entation results	Recoding Algorithm				
	Seidel [8] ß2^{8}	*Dimitrov [9]* ß'2^{8}	*Ours* ß"2^{8}	*Savings / ß2^{8} %*	*Savings / ß'2^{8} %*
Area[1]	5251	6599	3219	63	105
Energy[2]	2.49	2.48	1.63	53	52
Speed[3]	48.62	43.17	52.4	8	21

1: Area occupation in number of Virtex-6 slices
2: Energy consumption per multiplication operation (pJ)
3: Million multiplications per second (MMPS)

VI. NEW RADIX 2^{r} MULTI-PRECISION MULTIPLICATION TECHNIQUE

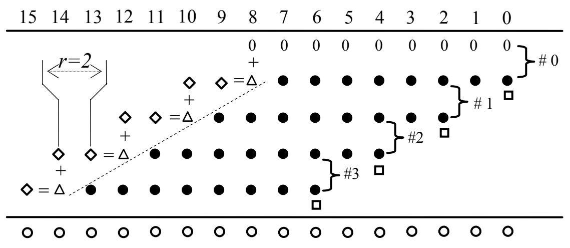
Prior to develop a highly-scalable multi-precision multiplier, the need for a flexible and low-power sign-extension technique is mandatory.

A. New radix 2^{r} sign extension technique

Though many low-power sign extension techniques exist in the literature, they are not adapted to reconfigurability. The reason for this shortcoming is that the correction bits must be calculated for each value of operand-size *n* [11][16]. Besides, to the authors' knowledge, no sign-extension solution exists for radix based multiplication (*r*). In what follows, we propose a generic low-power solution that circumvents these two obstacles. It is illustrated by Fig. 12 for *n*=8 and *r*=2, but can be systematically extended to any *n* and *r* values. Intuitively, we are not simultaneously performing the sum of the partial products, but each partial product of current step *j* is added to the sum of the preceding ones (from 0 to *j*-1). The rationale for the number of sign-bits to the left can be done locally, step by step, row by row. In other words, we have to take advantage of the fact that the partial sum already contains the sum of the sign bits of previous partial products. We must simply ensure that the sum output of the sign bit of current step *j* is added to the two most-significant bits of the next step (*j*+1). To generalize to radix ß2^{r} multiplication, the sign-bit (n^{th} position bit) of each partial product is extended with *r* bits to the left (*r*-1 for a maximum shift, plus one bit for the sign), and the sum output of the sign bit of step *j* is added to the *r* most-significant bits of the next step (*j*+1).

B. New Radix 2^{r} Multi-precision multiplication technique

In traditional *n*×*n* bit multi-precision multipliers, there is possibility to perform either a single *n*×*n* double precision, or a single *n*/2×*n*/2 simple precision, or a twin parallel *n*/2×*n*/2 simple precision multiplication. This is made possible by partitioning the two operands *X* and *Y* into respectively most and less significant sub-operands X_H Y_H, *and* X_L Y_L. A number of solutions exist and are summarized in [11][12]. Unfortunately, they are either restricted to unsigned multiplication, or they do not take power consumption into consideration, or they are not flexible enough. We propose hereafter a new technique that not only overcomes all above-mentioned shortcomings, but also allows a customized partitioning of the operands into any number of slices as well as in any slice sizes. Besides, this new technique is well adapted to radix based multiplication. Its features are compared to the technique presented in [11].



●: partial product bit; ○ : product bit; △: sign bit; ◇ sum of sign bits; □: negative one inserted into carry-in of the adder; #*i* : step *i*

Fig. 12. Low-power sign-extension technique for the particular case (*n, r*)=(8, 2)

TABLE VI
DELAY IN ADDER LEVELS FOR $r = 64$

Reduction Tree	Recoding Algorithm									
	ß2^{2}	ß2^{3}	ß2^{5}	ß2^{8}	ß'2^{8}	ß"2^{8}	ß2^{13}	ß2^{16}	ß2^{24}	ß2^{32}
Linear	32	23	15	14	9	12	14	14	18	22
Logarithmic	6*	7	7	7	5*	7	8	7	7	7

Total delay = PPG delay (d_i) + Reduction Tree Delay.
*: Note that d_2<<d'_8

Let us take equation (1) and apply it to *X* and *Y* for *r*=*n*/2, we obtain: $X = \sum_{j=0}^{1} Q_j 2^{\frac{n}{2}j} = Q_1 2^{\frac{n}{2}} + Q_0 = X_H + X_L$. Hence,

$$X.Y = Q_1 P_1 2^{n} + Q_1 P_0 2^{\frac{n}{2}} + Q_0 P_1 2^{\frac{n}{2}} + Q_0 P_0$$
$$= X_H.Y_H + X_H.Y_L + X_L.Y_H + X_L.Y_L \quad (13)$$

Note that Q_1 and Q_0 are (n/2)+1 bit size, but x_{-1} can be omitted from Q_0 since it is stuck at zero. Thus, we obtain four independent signed multipliers: $X_H.Y_H$, $X_H.Y_L$, $X_L.Y_H$, $X_L.Y_L$ which are respectively (*n*/2)+1×(*n*/2), (*n*/2)+1×*n*/2, *n*/2×(*n*/2), *n*/2×*n*/2 bit size. Fig. 13 illustrates the implementation of equation (13) for a signed 16x16 bit multiplier based on recoding algorithm ß2^{2} with *r*=2. Equation (13) eliminates the cumbersome term (EV×$2^{n/2}$) in equation (6) of [11] as well as the necessary logic for its generation. More importantly, in Fig. 13, four 8x8 bit multiplications can be performed simultaneously, whereas in [11] only two are allowed because of the shared terms

($EV \times 2^{n/2}$) and CV required for the sign extension. Without counting the necessary EV generation logic and the use of inverters for the negation of the sign bits, the partitioning proposed in [11] consumes a total bit count of 205 for a 16x16 bit multiplier, while ours requires 198 bits.

Note that equation (5) can be advantageously used to partition X_H *and* X_L with different bit lengths. For instance, with r=n, s=3n/4 and t=n/4, we obtain: $X = P + T\,2^{\frac{3n}{4}}$

$$\begin{aligned} X.Y &= P\,P' + P\,T'\,2^{\frac{3n}{4}} + T\,P'\,2^{\frac{3n}{4}} + T\,T'\,2^{\frac{3n}{2}} \\ &= X_L.Y_L + X_L.Y_H + X_H.Y_L + X_H.Y_H \qquad (14) \end{aligned}$$

Four independent signed multipliers are generated: $X_H.Y_H$, $X_H.Y_L$, $X_L.Y_H$, $X_L.Y_L$, which are respectively (n/4)+1×(n/4), (n/4)+1×(3n/4), (3n/4)+1×(n/4), and (3n/4)×(3n/4) bit size. The translation of equation (14) into architecture is depicted by Fig. 14. Both partitioning schemes (Fig. 13 and Fig 14) needs the same amount of bits (198).

More efficiently, equation (13) can be combined with ß"2^8 algorithm for the recoding of Y_H and Y_L sub-multiplicands to produce a faster partitioning (Fig. 15) for operand sizes larger than 16 bits according to the implementation results shown in Fig. 10.

More importantly, equation (1) can be used to partition the X and Y operands into any desired number of slices depending on r value. Choosing for instance r=n/4 results into the following partitioning:

$$X = \sum_{j=0}^{3} Q_j\,2^{\frac{n}{4}j} = Q_3\,2^{\frac{3n}{4}} + Q_2\,2^{\frac{n}{2}} + Q_1\,2^{\frac{n}{4}} + Q_0 \text{. Hence,}$$

$$\begin{aligned} X.Y = \; & Q_3P_3\,2^{\frac{3n}{2}} + Q_3P_2\,2^{\frac{5n}{4}} + Q_3P_1\,2^{n} + Q_3P_0\,2^{\frac{3n}{4}} \\ & + Q_2P_3\,2^{\frac{5n}{4}} + Q_2P_2\,2^{n} + Q_2P_1\,2^{\frac{3n}{4}} + Q_2P_0\,2^{\frac{n}{2}} \\ & + Q_1P_3\,2^{n} + Q_1P_2\,2^{\frac{3n}{4}} + Q_1P_1\,2^{\frac{n}{2}} + Q_1P_0\,2^{\frac{n}{4}} \\ & + Q_0P_3\,2^{\frac{3n}{4}} + Q_0P_2\,2^{\frac{n}{2}} + Q_0P_1\,2^{\frac{n}{4}} + Q_0P_0 \\ = \; & X_{HH}.Y_{HH} + X_{HH}.Y_{HL} + X_{HH}.Y_{LH} + X_{HH}.Y_{LL} \\ & + X_{HL}.Y_{HH} + X_{HL}.Y_{HL} + X_{HL}.Y_{LH} + X_{HL}.Y_{LL} \\ & + X_{LH}.Y_{HH} + X_{LH}.Y_{HL} + X_{LH}.Y_{LH} + X_{LH}.Y_{LL} \\ & + X_{LL}.Y_{HH} + X_{LL}.Y_{HL} + X_{LL}.Y_{LH} + X_{LL}.Y_{LL} \qquad (15) \end{aligned}$$

Equation (15) generates sixteen independent signed multipliers. All are (n/4)+1×(n/4) bit size, except $X_{LL}.Y_{HH}$, $X_{LL}.Y_{HL}$, $X_{LL}.Y_{LH}$, $X_{LL}.Y_{LL}$ which are (n/4)×(n/4) bit size. The implementation details of equation (15) for n=16 based on ß2^2 with r=2 are described in Fig. 16. Equation (15) requires a total bit count of 254 which induces an overhead of 28% compared to equation (13).

Finally, equation (1) and (5) can be combined with any proposed recoding algorithm (ß2^r) to produce any desired multi-precision multiplication scheme.

VII. Conclusion and Futur Work

We developed a recursive version of the multibit recoding multiplication algorithm which enabled to solve two hard problems: radix 2^r signed multiplication and radix 2^r multi-precision signed multiplication. The former is odd-multiple free solution with advanced capabilities for multiplication-intensive applications that must dissipate minimal power while operating at high speed. In addition, the solution is highly-scalable allowing a hardware-friendly partitioning that can be tailored to the desired performance and power budget.

We deliberately opted for FPGA implementation to rapidly explore a large number of variants of the recursive algorithm. Only ten recoding algorithms have been selected and reported in this paper. We first gave priority to a serial/parallel implementation as it is the most appropriate to designing embedded finite-word-length controllers, which is our ultimate objective. A fully-parallel implementation will be given the same attention for further investigation and optimization.

Guided by *Mux* and *Add* indicators, even higher odd-multiple free recoding algorithms (ß2^{64}, ß2^{128}, ß2^{256},…) can be generated to efficiently cope with large-operand-size applications, such as in cryptography. However, for large r values, the use of advanced optimization heuristics becomes mandatory in order to determine the primary radix (2^1, 2^2, 2^5, and 2^8) configuration that leads to the optimal implementation of the desired radix. This issue is being explored at present time and we plan to report our results in a forthcoming paper.

As for the multi-precision solution, this latter would not have been possible without the development of a flexible sign-extension technique. Based on the new recursive algorithm, we proposed a generic partitioning scheme that can be adapted to any size combination of the operands in order to reduce the power consumption while increasing the computational throughput. This new solution will be deeply explored for further optimizations using the proposed radix 2^r algorithms.

Appendix

A. Multibit Recoding Algorithm

Let X be an n-bit two's complement format binary integer. The value of X can then be found from:[3]

$$X = -x_{n-1}.2^{n-1} + \sum_{q=0}^{n-2} x_q.2q \qquad (2.1.1)$$

Note that this is a uniform representation for both positive and negative numbers. X is positive if x_{n-1} = 0 and is negative if x_{n-1} =1. An SD representation of X in radix 2^k ($k \geq 1$) will have n/k signed digits[4] $D_{n/k-1}, D_{n/k-2}, \ldots, D_1, D_o$. In this new radix ($2^k$) the value of x can be rewritten as

$$X = \sum_{i=0}^{n/k-1} D_i.2^{ki} \qquad (2.1.2)$$

where digits D_i are found from bits x_i of X according to

$$D_i = x_{ki-1} + \left(\sum_{j=1}^{k-1} x_{ki+j-1}.2^{j-1}\right) - x_{k(i+1)-1}.2^{k-1} \qquad (2.1.3)$$

And $$x_{-1} = 0 \qquad (2.1.4)$$

Acknowledgment

This work is supported by “Centre de Développement des Technologies Avancées, CDTA,” Algiers, Algeria, under project contract number: 21/CRSOC/DMN/CDTA/2011. The project progresses under a close cooperation with “Franche Comté Electronique Mécanique Thermique et Optique - Sciences et Technologies, FEMTO-ST ” Besançon, France.

The authors wish to thank D. Bouchaffra and B. Djezzar for their careful review of this manuscript.

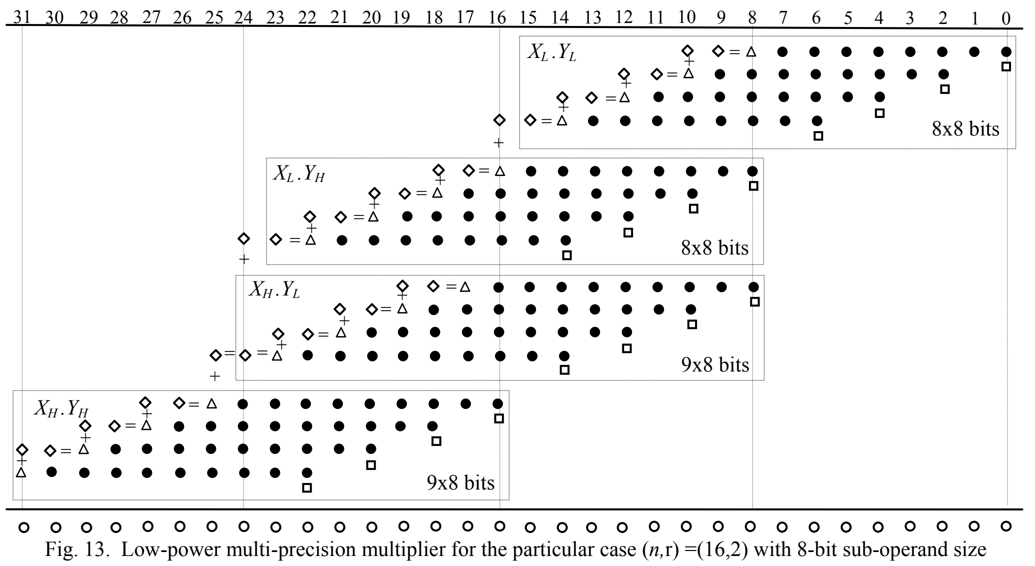


Fig. 13. Low-power multi-precision multiplier for the particular case (*n*,r) =(16,2) with 8-bit sub-operand size

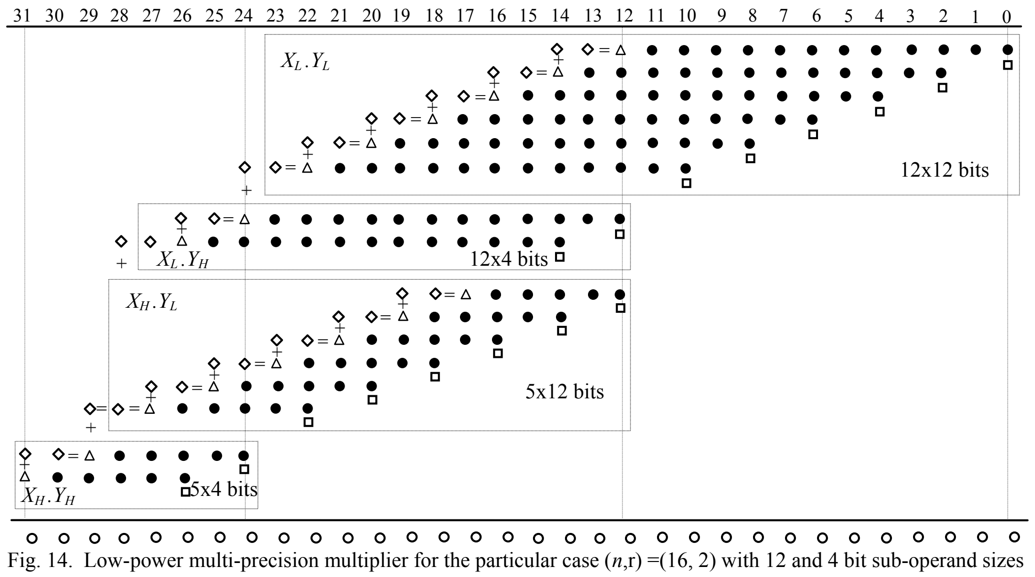


Fig. 14. Low-power multi-precision multiplier for the particular case (*n*,r) =(16, 2) with 12 and 4 bit sub-operand sizes

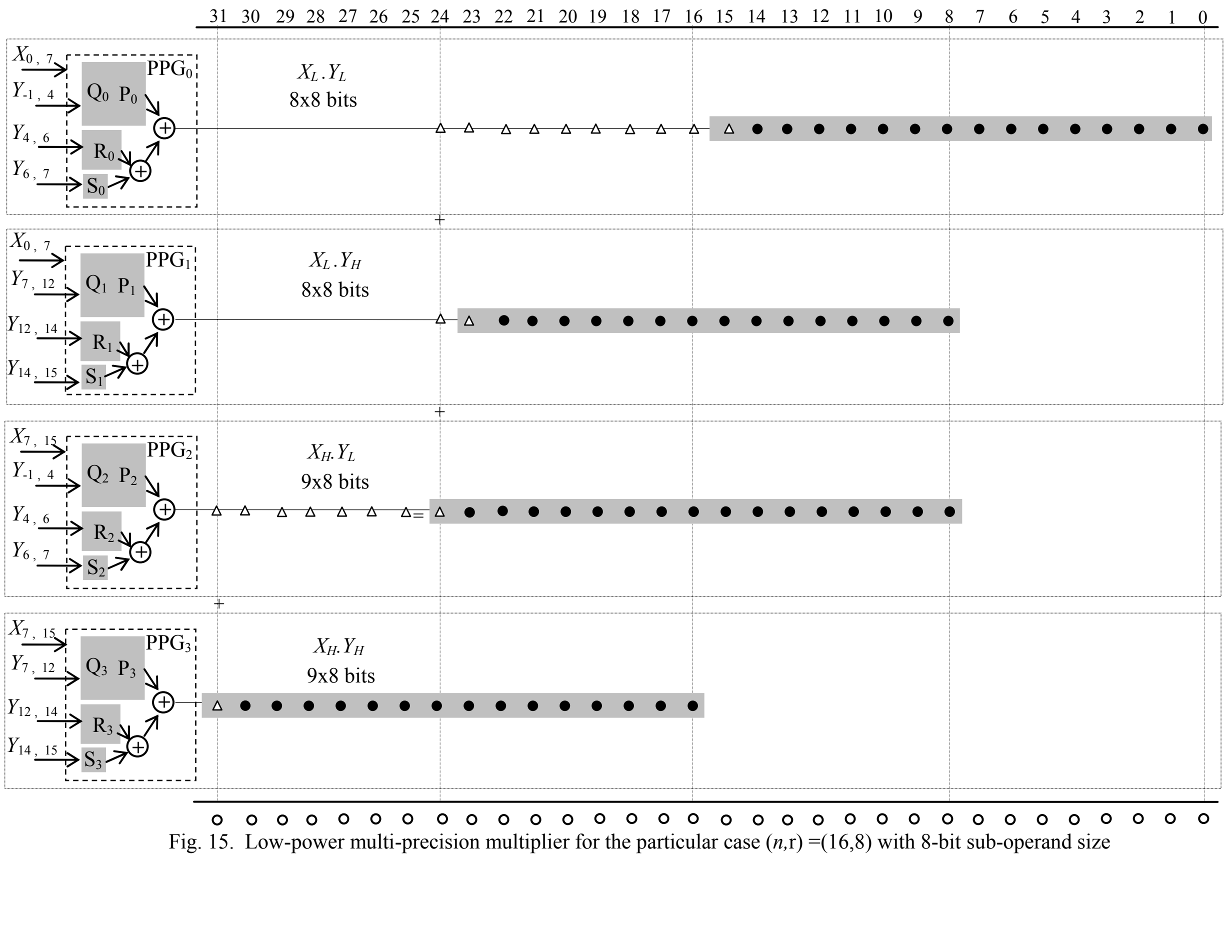


Fig. 15. Low-power multi-precision multiplier for the particular case (*n*,r) =(16,8) with 8-bit sub-operand size

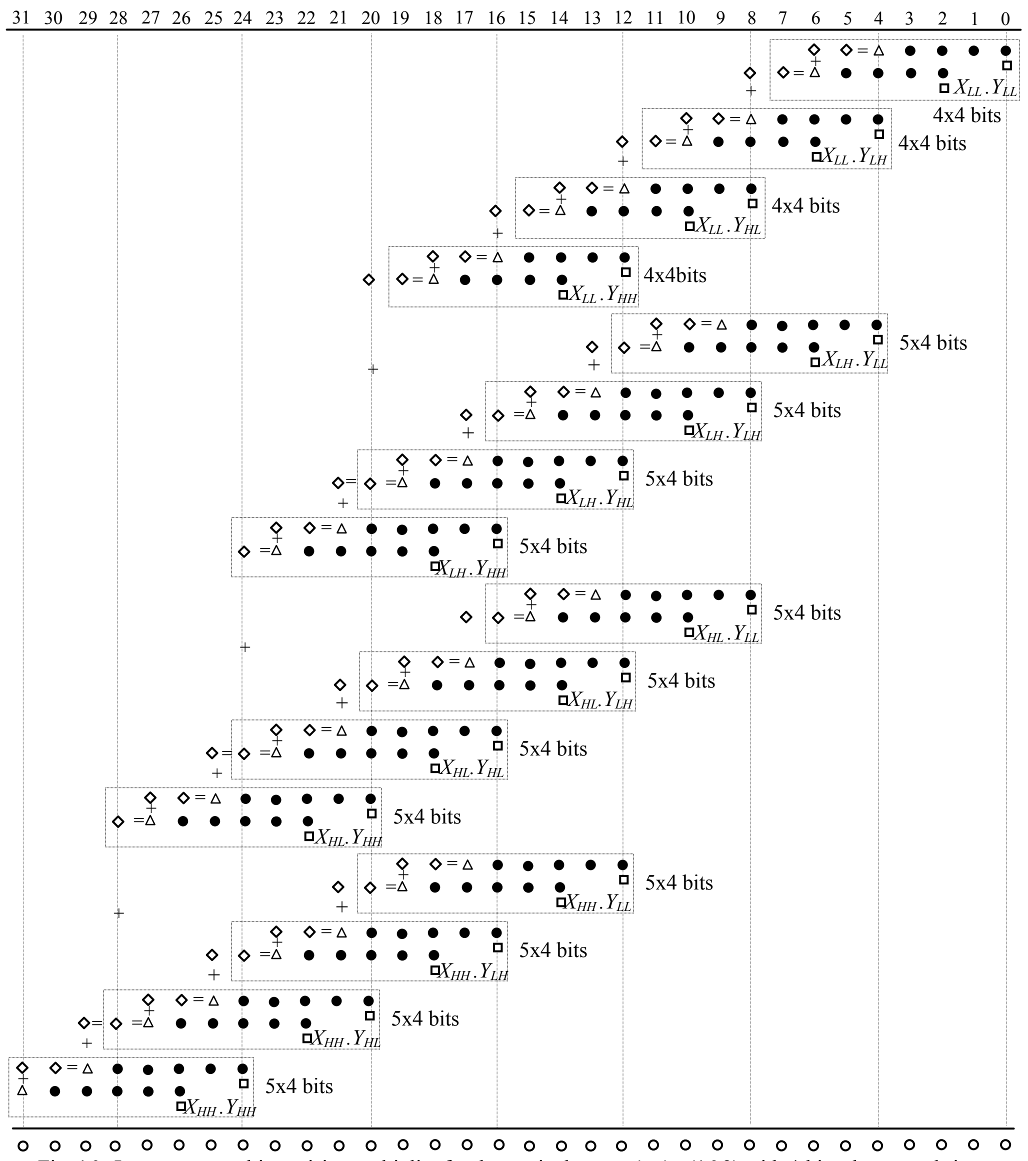


Fig. 16. Low-power multi-precision multiplier for the particular case (n,r) =(16,2) with 4-bit sub-operand size

References

[1] Reports on System Drivers of the International Technology Roadmap for Semiconductors (ITRS), 2009 and 2010. Available: www.itrs.net/reports.html

[2] O.L. MacSorley, "High-Speed Arithmetic in Binary Computers," Proceedings of the IRE, Vol. 49(1), pp. 67-91, January 1961.

[3] L. Dadda, "Some Schemes for Parallel Adders," Alta Frequenza, vol. 34, N° 5, pp. 349-356, May 1965.

[4] Z. Huang and M.D. Ercegovac, "Two-Dimensional Signal Gating for Low-Power Array Multiplier Design," Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS), pp. 489-492, May 2002.

[5] F. Lamberti, "Reducing the Computation Time in (Short Bit-Width) Two's Complement Multiplier," IEEE Trans. on Computers, vol. 60, N° 2, pp. 148-156, February 2011.

[6] S.R. Kuang, J.P. Wang, and C.Y. Guo, "Modified Booth Multipliers with a Regular Partial Product Array," IEEE Trans. on Circuit and Systems II, Express Brief, vol. 56, N° 5, May 2009.

[7] H. Sam, and A. Gupta, "A Generalized Multibit Recoding of Two's Complement Binary Numbers and its Proof with Application in Multiplier Implementation," IEEE Trans. on Computers, vol. 39, N° 8, August 1990.

[8] P.M. Seidel, L. D. McFearin, and D.W. Matula, "Secondary Radix Recodings for Higher Radix Multipliers," IEEE Trans. on Computers, vol. 54, N°2, February 2005.

[9] V.S. Dimitrov, K.U. Järvinen, and J. adikari, "Area Efficient Multipliers Based on Multiple-Radix Representations," IEEE Trans. on Computers, vol. 60, N° 2, pp 189-201, February 2011.

[10] A. D. Booth, "A Signed Binary Multiplication Te:chnique," Quarterly J. Mech. Appl. Math., Vol. 4, part 2, pp. 236-240,1951.

[11] S.R. Kuang, J.P. Wang, "Design of Power-Efficient Configurable Booth Multiplier," IEEE Trans. on Circuit and Systems I, vol. 57, N° 3, March 2010.

[12] M. Själander and P. Larsson-Edefors, "Multiplication Acceleration Through Twin Precision," IEEE Trans. on Very Large Scale Integration (VLSI) Systems, Vol. 17, N° 9, September 2009.

[13] P.M. Seidel, L. D. McFearin, and D.W. Matula, "Binary Multiplication Radix-32 and Radix-256," Proceedings of the IEEE Symposium on Computer Arithmetic (ARITH-15), ISBN: 0-7695-1150-3, pp. 23-32, USA, June 2001.

[14] A.K. Oudjida, N. Chaillet, A. Liacha, M. Hamerlain, and M.L. Berrandjia, "High-Speed and Low-Power PID Structures for Embedded Applications" Proceedings of the 21th edition of the International Workshop on Power and Timing Modeling, Optimization and Simulation PATMOS, LNCS 6951, pp. 257-266, Springer-Verlag Editor. Madrid, Spain, Sep. 26-29 2011.

[15] E. Manmasson et al., "FPGA in Industrial Control Applications," IEEE Trans. on Industrial Informatics, vol. 7, N° 2, May 2011.

[16] M. Annaratone, "Digital CMOS Circuit Design," Kluwer Academic Publisher, 1986.