Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2009, Article ID 145130, 11 pages
doi:10.1155/2009/145130

Research Article

Efficient Scheme for Implementing Large Size Signed Multipliers Using Multigranular Embedded DSP Blocks in FPGAs

Shuli Gao, Dhamin Al-Khalili, and Noureddine Chabini

Department of Electrical and Computer Engineering, Royal Military College of Canada, ON, Canada K7K 7B4

Correspondence should be addressed to Dhamin Al-Khalili, alkhalili-d@rmc.ca

Received 21 August 2008; Accepted 27 January 2009

Recommended by Christophe Bobda

Modern FPGAs contain embedded DSP blocks, which can be configured as multipliers with more than one possible size. FPGA-based designs using these multigranular embedded blocks become more challenging when high speed and reduced area utilization are required. This paper proposes an efficient design methodology for implementing large size signed multipliers using multigranular small embedded blocks. The proposed approach has been implemented and tested targeting Altera's Stratix II FPGAs with the aid of the Quartus II software tool. The implementations of the multipliers have been carried out for operands with sizes ranging from 40 to 256 bits. Experimental results demonstrated that our design approach has outperformed the standard scheme used by Quartus II tool in terms of speed and area. On average, the delay reduction is about 20.7% and the area saving, in terms of ALUTs, is about 67.6%.

1. Introduction

Nowadays modern FPGAs offer highly sophisticated resources in the form of embedded blocks. These blocks vary in complexity from small size multipliers to core processors. Some of these blocks offer high degree of flexibility to cover a wide range of applications. A typical example is the DSP blocks in Altera's FPGA. These blocks can be configured to operate as 9×9-, 18×18-, or 36×36-bit multipliers [1]. This flexibility of operating as multigranular embedded blocks can be used to develop optimized realizations of large size computing functions, such as large size multiplications.

Arithmetic computations are needed in a wide range of applications and products. Some of these arithmetic computations deal with large size operands. Typical applications include scientific computation, cryptography, and data intensive systems. For instance, in climate modeling and computational physics, high-precision floating point processing is needed [2, 3], and these in turn require large operand fixed point multipliers. Another application which requires large size multiplications is processing-in-memory system used in data intensive multimedia and video applications [4].

There are various techniques presented in the literature which deal with efficient realization of signed array multiplication through optimization of partial product generation and partial product addition. The focus primarily is to reduce the delay of the critical path, and sometimes to meet other design objectives such as power dissipation and chip area [5–7]. Most of these techniques are based on bit level, hence suitable for ASIC or custom implementation. To map the same algorithms on LUTs in FPGAs, the realization becomes fairly inefficient, due to the interconnect delay and the generic nature of the LUTs. With the availability of highly optimized embedded multiplier blocks and incorporation of microarithmetic operation within the LUTs, the strategy for realization of multipliers is changed. For instance, an attempt has been made to realize the multipliers by utilizing 3 : 2 compressors provided by the 6-LUTs on Altera's FPGAs [1, 8]. This technique has achieved satisfactory results, and sometimes outperformed implementations based on the embedded multipliers. Another scheme has proposed a hybrid approach which utilizes embedded blocks and LUTs [9]. Both of these two techniques are effective when dealing with small size multipliers, however, large size multipliers require the use of highly efficient structures and to minimize

interconnect delays. This is only achievable through the use of the embedded multipliers or DSP blocks, which are available nowadays on myriad of FPGAs offered by vendors such as Xilinx and Altera.

For large size multiplication using FPGA devices, known algorithms normally segment the input operands based on single size embedded blocks [10, 11]. For the case of Xilinx' FPGAs for instance, a sign-extension-based approach [12] is used for realizing the large size multiplications, which employs 18×18-bit embedded signed multipliers as the basic blocks. For Altera's FPGAs, a standard approach is to use single decomposition to implement the large size multipliers [1], which is also based on the 18×18 embedded multipliers. However, since the embedded DSP blocks in newer Altera's FPGA devices, such as Stratix II and later, can be configured as 9×9-bit, 18×18-bit, and 36×36-bit multipliers, then it is possible to use multiple size embedded blocks as the basic units to efficiently implement large size multipliers.

In our previous work, we developed an efficient design approach for the implementation of large size unsigned multipliers [13]. A more systematic approach was presented in [14] with a set of design rules for addition of partial products leading to more efficient realization. A structured methodology was then developed to implement large size 2's complement multiplier based on Baugh-Wooley algorithm. Taking advantage of the multigranularity of the embedded DSP blocks, the authors proposed a scheme to design highly efficient 256×256 2's complement multiplier using new approach for sign extension [15]. In this paper, we present a design scheme for the general case to realize large size signed multipliers based on multiple size embedded blocks. We propose a divide-and-conquer-based strategy with a multilevel decomposition procedure, followed by an optimized approach for realizing the required additions of the partial products to obtain the final result. We have also dealt with special cases so that more improvements can be achieved for a range of input operands from 40 to 284 bits.

The remainder of this paper is organized as follows. Section 2 describes the architecture of large size multipliers and a new sign-extension scheme used in this paper. Section 3 presents the proposed design approach of large size multigranular-block-based signed multipliers, and a design example for a 256×256-bit multiplier is provided in Section 4. In Section 5, experimental results and comparisons are presented. Finally, conclusions are given in Section 6.

2. Implementation of Large Size Multipliers Based on Single Size Embedded Blocks

In this section, we describe the decomposition method of large size multiplications based on single size embedded blocks, and a new sign-extension scheme to sum the generated partial products.

2.1. Architecture of Single-Size-Embedded-Block-Based Large Size Multipliers. To implement a large size multiplication using single size embedded multipliers in FPGAs, the input operands are decomposed based on the size of the embedded blocks [14]. Assuming that the size of each 2's complement embedded block is n bits, each input operand is decomposed into m segments with $m = \lceil k/(n-1)\rceil$, where k is the size of the large size multiplier, and $\lceil z\rceil$ is the ceil function of z. The expressions of the operands are represented as follows:

$$X = X_{m-1}\times 2^{(m-1)\times(n-1)} + \sum_{i=0}^{m-2} X_i \times \left(2^{(n-1)}\right)^i,$$
$$Y = Y_{m-1}\times 2^{(m-1)\times(n-1)} + \sum_{i=0}^{m-2} Y_i \times \left(2^{(n-1)}\right)^i. \tag{1}$$

In (1), the segment X_i or Y_i is $(n-1)$-bit positive number for $0 \le i < (m-1)$. For $i = (m-1)$, the segment X_{m-1} or Y_{m-1} is $(k-(m-1)\times(n-1))$-bit signed number, which is in the range of 2 to n bits.

By multiplying the segmented inputs presented in (1), the output of the multiplier is expressed as

$$\begin{aligned} Z &= X\times Y \\ &= \left[X_{m-1}\times 2^{(m-1)\times(n-1)} + \sum_{i=0}^{m-2} X_i \times \left(2^{(n-1)}\right)^i\right] \\ &\quad\times \left[Y_{m-1}\times 2^{(m-1)\times(n-1)} + \sum_{i=0}^{m-2} Y_i \times \left(2^{(n-1)}\right)^i\right] \\ &= \left[(X_{m-1}\times Y_{m-1})\times 2^{2(m-1)\times(n-1)}\right] \\ &\quad+\left[X_{m-1}\times 2^{(m-1)\times(n-1)} \times \sum_{i=0}^{m-2} Y_i \times \left(2^{(n-1)}\right)^i\right] \\ &\quad+\left[Y_{m-1}\times 2^{(m-1)\times(n-1)} \times \sum_{i=0}^{m-2} X_i \times \left(2^{(n-1)}\right)^i\right] \\ &\quad+\left[\sum_{i=0}^{m-2} X_i \times \left(2^{(n-1)}\right)^i \times \sum_{i=0}^{m-2} Y_i \times \left(2^{(n-1)}\right)^i\right]. \end{aligned} \tag{2}$$

According to the optimized design approach of large size multipliers proposed in [14], all partial products in (2) can be organized as shown in Figure 1, where $(n-1)$ is denoted as n'.

After all operands shown in Figure 1 are achieved, multiple level additions are required for adding all these 2's complement operands. To reduce the area and the execution delay, a set of optimization design rules can be followed, which is proposed in [14].

2.2. New Sign-Extension Scheme for Large Size Signed Multipliers. To save area and reduce the execution delay, our new sign-extension scheme proposed in [15] first organizes the additions following the set of design rules, and then extends the sign bits according to the resulting organized additions. For example, the first level addition of the large size multiplication is to add each pair of operands that have the same size as shown in Figure 1. The sign extension requires only one bit to take care of the carry of the addition.

After the first level addition, all operands to be added further are 2's complement and have different sizes. Then,

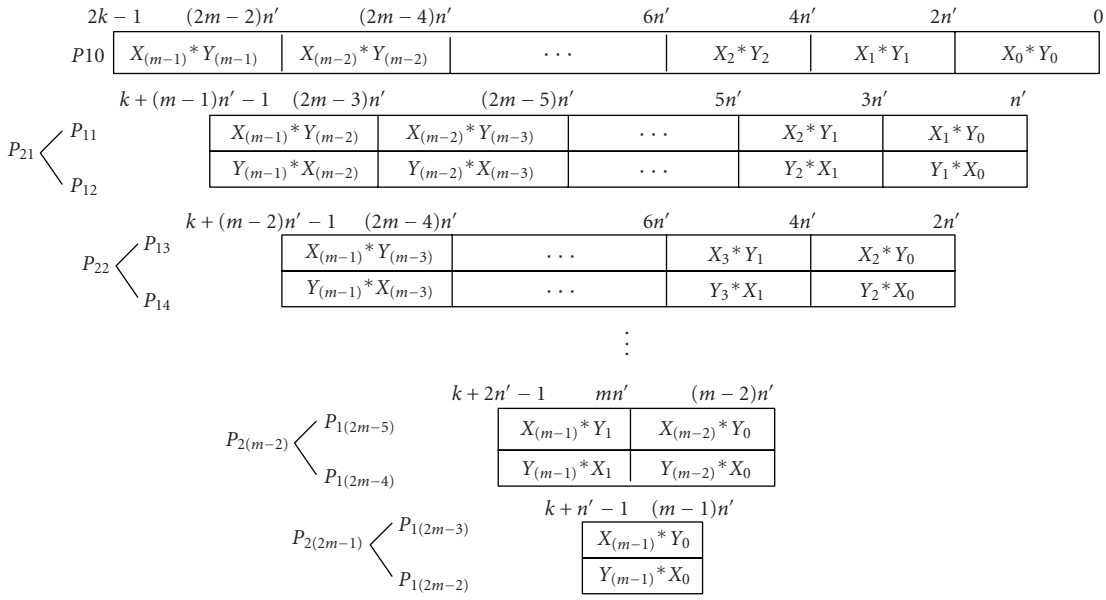


Figure 1: Organization of the partial products of large size signed multiplier.

at the second level and subsequent levels of addition, our proposed new sign-extension scheme extends the sign bits of the larger size operand, as shown in Figure 2, by one bits, and sign extend of the smaller size operands to the same position as that of the larger one. Moreover, to reduce the size of adders, the least significant bits that do not overlap with the other operand are concatenated to the output of the adder.

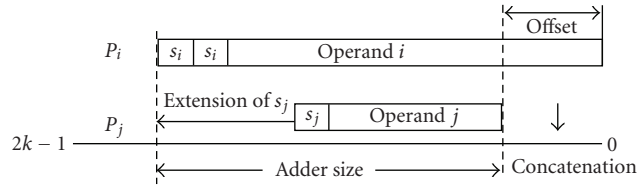


Figure 2: Structure of the adders in the new sign-extension scheme.

3. Design of Large Size Signed Multipliers Using Multigranular Embedded Blocks

In this section, we describe our proposed multilevel decomposition approach for the implementation of large size signed multiplications using multigranular embedded multipliers.

3.1. Decomposition of Large Size Multipliers Based on Multigranular Embedded Blocks. We assume that multigranularity is based on three types of building blocks. They are of different bit widths: n, t, and p, where

$$t = \frac{n}{2}, \qquad p = \frac{t}{2}. \tag{3}$$

For Altera's FPGAs, for instance, $n = 36$, $t = 18$, and $p = 9$.

To optimize the design of the large size multipliers, the decomposition is first processed based on the largest size building blocks. Figure 3 illustrates the decomposition of the multiplication, where X and Y are the input operands of the multiplier to be implemented.

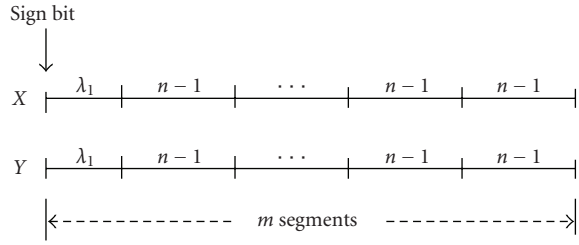


Figure 3: Decomposition based on the largest size of embedded blocks.

By multiplying the segmented inputs, three types of multipliers are required to generate the partial products. They are

$$\begin{aligned} &(n-1)\times(n-1)\text{-bit unsigned multiplier},\\ &(\lambda_1)\times(\lambda_1)\text{-bit signed multiplier},\\ &(\lambda_1)\times(n-1)\text{-bit signed multiplier}. \end{aligned} \tag{4}$$

The first type of multiplication can be implemented using $n \times n$-bit embedded signed multiplier with the sign bit forced to zero.

The second type of multiplication, $(\lambda_1) \times (\lambda_1)$-bit signed multiplier required only once, can be implemented using one

of $n \times n$, $t \times t$ or $p \times p$-bit embedded signed multiplier according to the value of λ_1. If λ_1 is less than or equal to p bits, then a $p \times p$-bit embedded multiplieris used; if λ_1 is greater than p but less than or equal to t bits, then a $t \times t$-bit embedded multiplieris required; otherwise, an $n \times n$-bit embedded multiplier is needed.

The last type of multiplication is $(\lambda_1) \times (n-1)$-bit signed multiplier. To efficiently implement this kind of multiplier, smaller size embedded blocks, such as $t \times t$-bit embedded multipliers, are utilized. There are two scenarios to be considered. Equation (5) illustrates these two situations:

$$\begin{aligned} 1 < \lambda_1 &\le [t-(m-1)], \\ [t-(m-1)] < \lambda_1 &\le n, \end{aligned} \tag{5}$$

where $n = 36$ and $t = 18$. Table 1 categorizes multipliers for bit sizes from 37 to 281 for these two situations with segments m from 2 to 8.

In Range 1, smaller size embedded multipliers can be used for the implementation based on double-level decomposition. To do this, the size of the first level decomposition is based on $(n-2)$ bits instead of $(n-1)$ bits since it needs to be decomposed further as two subsegments. Figure 4 illustrates the first level decomposition for the size in Range 1. For example, a 120×120-bit multiplier, which is one of the cases in Range 1, can be decomposed into four segments of 18, 34, 34, and 34 bits.

The second level decomposition is to separate each 34-bit operand as two 17 bits. Thus, the 18×34-bit multiplication can be implemented using 18×18-bit embedded multipliers.

On the other hand, the cases in Range 2, double level decomposition will not lead to optimized solution since the size of the most significant segment, λ_2 in this case, is more than t bits. For example, for a 121×121-bit multiplier, if the 121-bit operand first is decomposed as 19, 34, 34, and 34, the sizes of all operands in this multiplication are greater than t bits, so the $t \times t$-bit embedded multiplier cannot be used. Therefore, only single-level decomposition is needed and $n \times n$-bit embedded multipliers are required. The block size for this decomposition is equal to $(n-1) = 35$ bits. For this example, the 121-bit operand is decomposed into 4 segments of 16, 35, 35, and 35 bits.

3.2. Implementation of Large Size Multipliers Based on Multigranular Embedded Multipliers. In this section, we will describe the implementation approaches for the realization of large size multipliers for the scenarios presented in the Section 3.1.

3.2.1. Implementation of Large Size Multipliers Based on Double Decomposition. Double decomposition is used for the bit size located in Range 1. The first level decomposition is to decompose each input operand, X or Y, into $\lceil k/(n-2) \rceil = m$ segments, where k is the size of the multiplication to be implemented, m is the number of segments, and $n = 36$ is the size of the largest embedded multiplier. After the first level decomposition, the segmented input operands are multiplied and the partial products are organized as shown in Figure 1. The partial product $X_{m-1} \times Y_{m-1}$ can

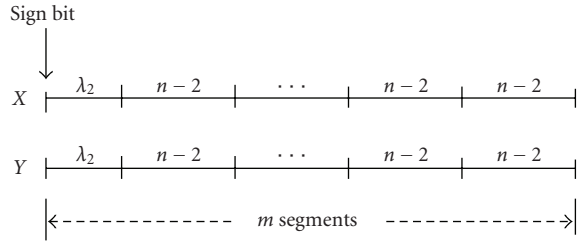


Figure 4: First level decomposition for a large size multiplication in Range 1.

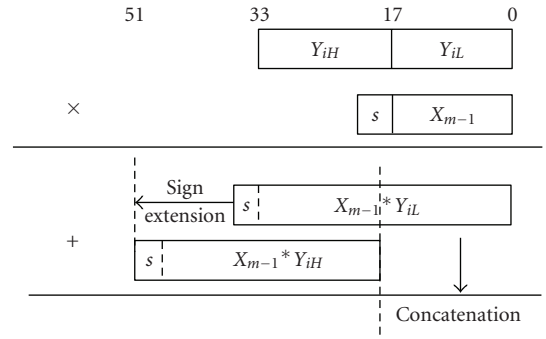


Figure 5: 18×34-bit multiplication using 18×18-bit embedded blocks.

be implemented by $t \times t$-bit embedded multipliers if the size of X_{m-1} or Y_{m-1} is greater than p bits, or by $p \times p$-bit embedded multipliers if it is equal to or less than p bits. The partial product $X_i \times Y_j$ $(i, j = 0, 1, \ldots, m-2)$ with the size of $(n-2)$ bits, and it can be implemented using the $n \times n$-bit embedded multipliers with the most two significant bits forced to zeros. The last partial product, $X_{m-1} \times Y_j$, or $Y_{m-1} \times X_i$ $(i, j = 0, 1, \ldots, m-2)$, has $(n-2)$ bits in X_i or Y_j, and less than or equal to t bits in X_{m-1} or Y_{m-1}. Then, a second level decomposition is performed. The $(n-2)$-bit operand is decomposed as two subsegments with $(t-1)$ bits each. After the second level decomposition, the segmented multiplication, $(X_{m-1} \times Y_j)$ or $(Y_{m-1} \times X_i)$, is implemented using $t \times t$ embedded multipliers. Figure 5 presents an example for carrying out this multiplication. This process requires two $t \times t$-bit embedded signed multipliers and one adder. Sign-extension is performed before the addition. Also, the concatenation operation is used for the last $(t-1) = 17$ bits of the partial products to reduce the size of the adder. To use signed embedded multipliers for unsigned numbers, the sign bits of the embedded multipliers for the $(t-1)$-bit operands are forced to zeros.

Once all the segmented partial products are generated, the required additions can be performed following the design rules presented in [14].

3.2.2. Implementation of Large Size Multipliers Based on Single Decomposition. In the case of Range 2, single decomposition is performed since the most significant segment of the input operands contains more than t bits. In this case, the decomposition is based on $(n-1)$ bits, where n is the largest

Table 1: Decomposition of input bit sizes for large size multipliers.

Segments	Range 1: double decomposition $\{2 \le \lambda_1 \le [t-(m-1)]\} + (m-1) \times (n-1)$	Range 2: single decomposition $\{[(t+1)-(m-1)] \le \lambda_1 \le n\} + (m-1) \times (n-1)$
$m = 2$	37 to 52	53 to 71
$m = 3$	72 to 86	87 to 106
$m = 4$	107 to 120	121 to 141
$m = 5$	142 to 154	155 to 176
$m = 6$	177 to 188	189 to 211
$m = 7$	212 to 222	223 to 246
$m = 8$	247 to 256	257 to 281

Table 2: The ranges of input bit size with special cases for decompositions.

Range/segments	Range 1 $[40 + 35 \times (m-2)]$ to $[18 + 34 \times (m-1)]$	Special cases of Range 1 $[19 + 34 \times (m-1)]$ to $[21 + 34 \times (m-1)]$	Range 2 $[22 + 34 \times (m-1)]$ to $[36 + 35 \times (m-1)]$	Special cases of Range 2 $[37 + 35 \times (m-1)]$ to $[39 + 35 \times (m-1)]$
$m = 2$	40 to 52	53 to 55	56 to71	72 to 74
$m = 3$	75 to 86	87 to 89	90 to 106	107 to 109
$m = 4$	110 to 120	121 to 123	124 to 141	142 to 144
$m = 5$	145 to 154	155 to 157	158 to 176	177 to 179
$m = 6$	180 to 188	189 to 191	192 to 211	212 to 214
$m = 7$	215 to 222	223 to 225	226 to 246	247 to 249
$m = 8$	250 to 256	257 to 259	260 to 281	282 to 284

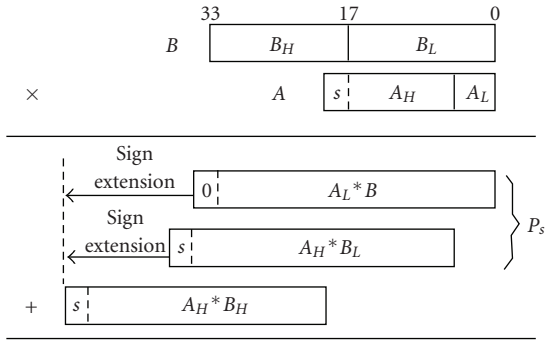


Figure 6: Generation of the partial product for the special case of Range 1.

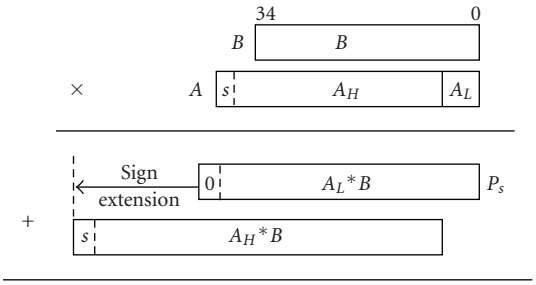


Figure 7: Gereration of the partial product for special cases of Range 2.

size of embedded blocks. After the decomposition, all partial products can be organized in the same way as shown in Figure 1 with $n = 36$. The optimized addition operations for summing these partial products are performed also based on the design rules summarized in [14].

3.3. Special Cases of the Implementation for the Large Size Multiplier. The special cases are referred to the multiplications such that the most significant segment of each operand has $r + t$ or $r + n$ bits, where r is in the range of 1 to 3 bits. For these special cases, the realizations of the multiplications, which involve these small size segments, are implemented using (Look Up Tables) LUTs instead of using embedded blocks. Based on our experimental analysis, for the cases with r greater than 3, the use of LUTs will result in larger delay and area utilization. Table 2 lists the special cases of Range 1 and Range 2. In the following, we will explain the algorithms for the designs of the special cases. The focus of these algorithms is to reduce the number of embedded blocks required in the designs.

3.3.1. Design of Special Cases of Range 1. In this special case, the most significant segment, X_{m-1} or Y_{m-1} is a signed number with a bit size of $t+1$ to $t+3$, and the other segments are positive of size of $(n-2)$. To explain the algorithm for this special case, let's assume that the signed segment, X_{m-1} or Y_{m-1}, is referred to as the A operand, and the other positive segment is referred to as the B operand. The A operand is decomposed as A_H and A_L. A_H has the most significant t bits of A, and A_L has the rest of the bits of A, which is 1 to 3 bits. The B operandis decomposed also to two segments with $(n-2)/2 = (t-1)$ bits each. The algorithm for the special

Case when the size of A_L is 1 bit
If $A_L = 0$ *then*
$P_S = A_H \times B_L \times 2^1$
Else
$P_S = A_H \times B_L \times 2^1 + B$
End if
$A \times B = A_H \times B_H \times 2^t + P_S$
Case when the size of A_L is 2 bits
If $A_L = 00$ *then*
$P_S = A_H \times B_L \times 2^2$
Else if $A_L = 01$ *then*
$P_S = A_H \times B_L \times 2^2 + B$
Else if $A_L = 10$ *then*
$P_S = A_H \times B_L \times 2^2 + B \times 2^1$
Else
$P_S = A_H \times B_L \times 2^2 + B \times 2^1 + B$
End if
$A \times B = A_H \times B_H \times 2^{(t+1)} + Ps$
Case when the size of A_L is 3 bits
If $A_L = 000$ *then*
$P_S = A_H \times B_L \times 2^3$
Else if $A_L = 001$ *then*
$P_S = A_H \times B_L \times 2^3 + B$
Else if $A_L = 010$ *then*
$P_S = A_H \times B_L \times 2^3 + B \times 2^1$
Else if $A_L = 011$ *then*
$P_S = A_H \times B_L \times 2^3 + B \times 2^1 + B$
Else if $A_L = 100$ *then*
$P_S = A_H \times B_L \times 2^3 + B \times 2^2$
Else if $A_L = 101$ *then*
$P_S = A_H \times B_L \times 2^3 + B \times 2^2 + B$
Else if $A_L = 110$ *then*
$P_S = A_H \times B_L \times 2^3 + B \times 2^2 + B \times 2^1$
Else
$P_S = A_H \times B_L \times 2^3 + B \times 2^2 + B \times 2^1 + B$
End if
$A \times B = A_H \times B_H \times 2^{(t+2)} + Ps$

Algorithm 1

Case when the size of A_L is 1 bit
If $A_L = 0$ *then*
$P_S = 0$
Else
$P_S = B$
End if
$A \times B = A_H \times B \times 2^1 + PS$
Case when the size of A_L is 2 bits
If $A_L = 00$ *then*
$P_S = 0$
Else if $A_L = 01$ *then*
$P_S = B$
Else if $A_L = 10$ *then*
$P_S = B \times 2^1$
Else
$P_S = B \times 2^1 + B$
End if
$A \times B = A_H \times B \times 2^2 + Ps$
Case when the size of A_L is 3 bits
If $A_L = 000$ *then*
$P_S = 0$
Else if $A_L = 001$ *then*
$P_S = B$
Else if $A_L = 010$ *then*
$P_S = B \times 2^1$
Else if $A_L = 011$ *then*
$P_S = B \times 2^1 + B$
Else if $A_L = 100$ *then*
$P_S = B \times 2^2$
Else if $A_L = 101$ *then*
$P_S = B \times 2^2 + B$
Else if $A_L = 110$ *then*
$P_S = B \times 2^2 + B \times 2^1$
Else
$P_S = B \times 2^2 + B \times 2^1 + B$
End if
$A \times B = A_H \times B \times 2^3 + Ps$

Algorithm 2

cases of Range 1 is graphically illustrated in Figure 6 and the pseudocode is given below and named as Algorithm 1. In this case, two 18×18-bit embedded multipliers are required instead of one 36×36-bit embedded block.

In addition, Algorithm 1 can be extended to the partial product, $X_{m-1} \times Y_{m-1}$. Since both operands, X_{m-1} or Y_{m-1}, have $t + 1$, $t + 2$ or $t + 3$ bits, it can be decomposed as two segments. One segment has the most significant t bits and the other segment has the rest of the bits, 1, 2 or 3, respectively. This multiplication is implemented by one $t \times t$-bit embedded multiplier with some additions rather than one $n \times n$-bit embedded multiplier.

3.3.2. Design of the Special Cases of Range 2. For these cases, the size of the most significant segments of the input operands, X_{m-1} and Y_{m-1}, is $(n + 1)$, $(n + 2)$, or $(n + 3)$ bits. These segmented operands are multiplied with the other segments that have a size of $(n - 1)$ bits.

We also let A denotes X_{m-1}or Y_{m-1}, and B is any other positive number segmented with $(n - 1)$ bits. The operand A is decomposed as A_H and A_L. A_H has the most significant n bits of A, and A_L has the rest, 1 to 3 bits. The multiplication of these two operands is illustrated in Figure 7, and the realization is based on the pseudocode given below and named as Algorithm 2.

In the special cases of Range 2, the partial product $X_{m-1} \times Y_{m-1}$ can be implemented in a similar way as it is done for Range 1. In this case, only one $n \times n$-bit embedded multiplier with some additions is required.

4. Design Example: A 256 × 256-bit Multigranular Signed Multiplier

As a design example, this section summarizes the implementation of a 256 × 256-bit signed multiplier using

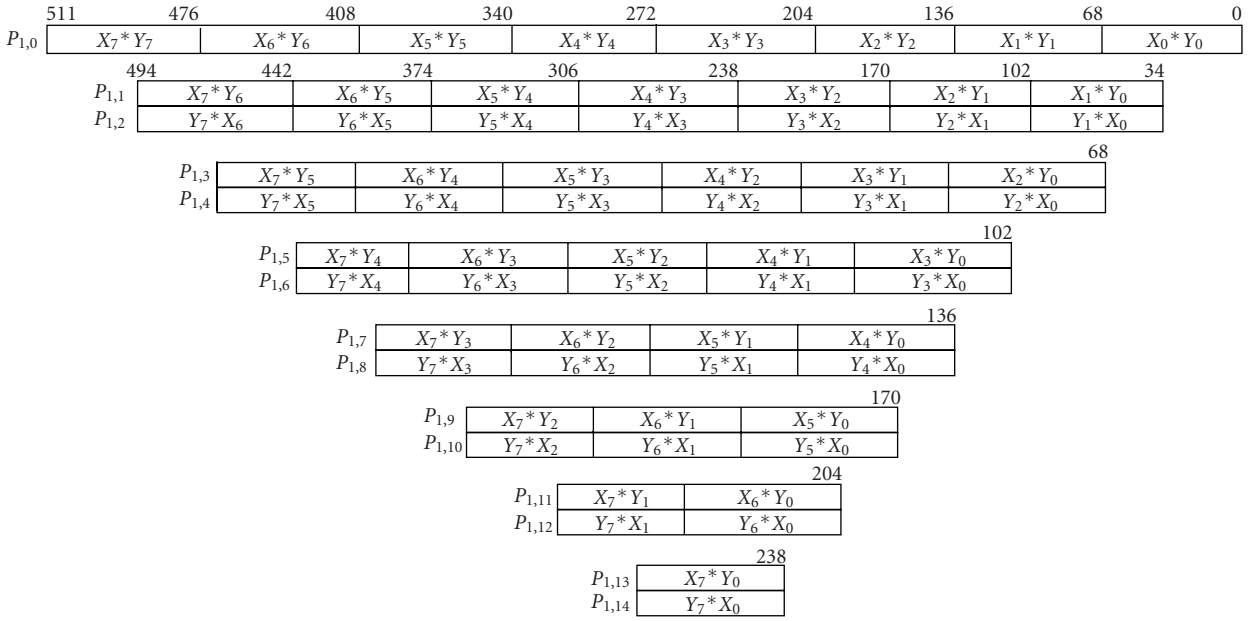


Figure 8: The segmented partial products of a 256 × 256-bit multiplier.

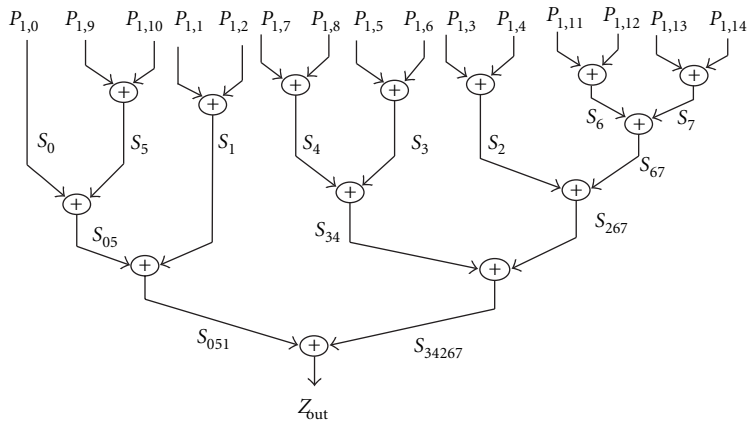


Figure 9: Flow digram of the additions required for a 256 × 256-bit multiplier.

multigranular embedded blocks with $n = 36$, $t = 18$, and $p = 9$. This design example is described with the following four steps.

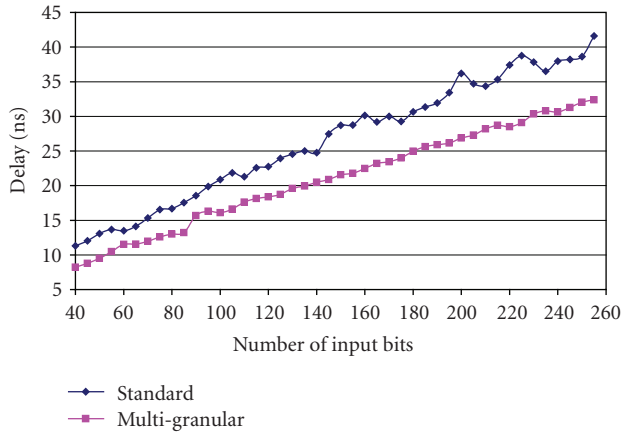
Step 1 (First level decomposition). Since the size of the operands is in Range 1, two levels of decomposition are required. The first level decomposition is based on $(n - 2) = 34$ bits. In this case, each of the 256-bit input operands is decomposed from the right- to left-hand side with 34 bits each. Because of $\lceil 256/34 \rceil = 8$ and $256 - 7 \times 34 = 18$, eight segments are required and the most significant segment has 18 bits. Since all the segmented operands have more than 9 bits, the 9 × 9-bit embedded blocks cannot be used in this case.

Step 2 (Generation of the segmented partial products). After the first-level decomposition, these segmented input operands are multiplied and the organization of all the partial products is shown in Figure 8.

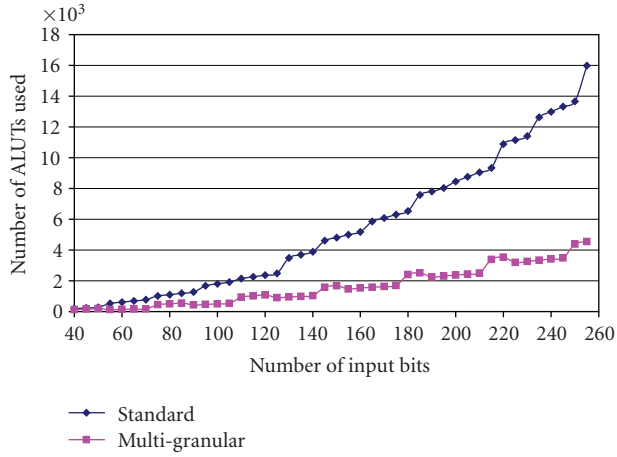
Step 3 (Second level decomposition). In Figure 8, two types of basic multipliers are required. One is 34 × 34-bit unsigned multipliers and the other is 18 × 34-bit signed multipliers. The 34 × 34-bit unsigned multiplication is implemented using 36 × 36-bit signed embedded multiplier with the first two bits forced to zeros, and the 18 × 34-bit multiplication has to be decomposed again.

At second level decomposition, each 34-bit operand is split into 17 bits each. Then, the 18 × 34-bit multiplication is implemented using the process shown in Figure 5. Also, the sign bits of the embedded multipliers for the 17-bit operands are forced to zeros.

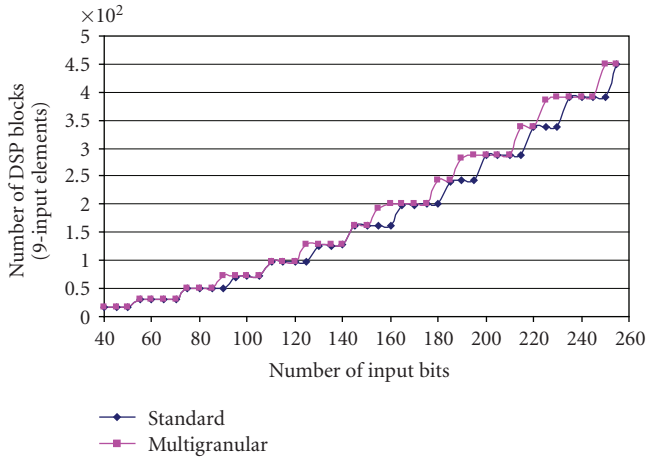
In this implementation, one 18 × 18-bit, forty nine 34 × 34-bit and fourteen 18 × 34-bit multipliers are



(a) Delay

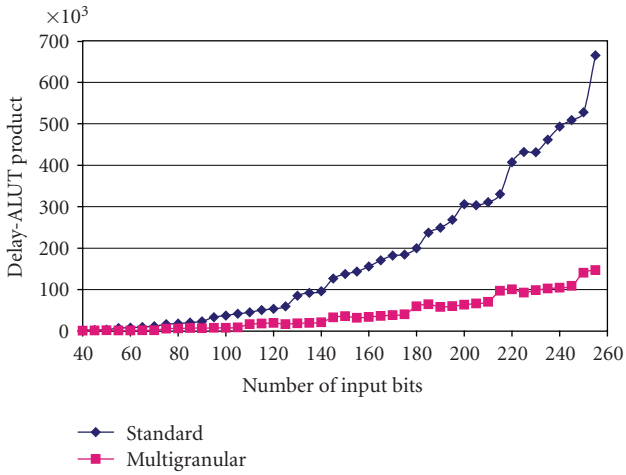


(b) Area

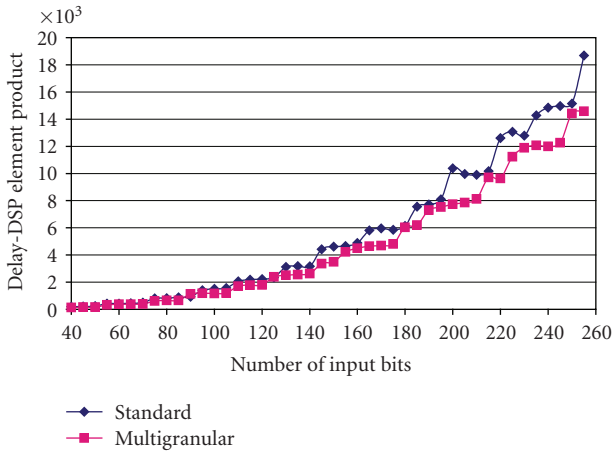


(c) DSP blocks

Figure 10: Comparison of the results.



(a) Delay-ALUT product



(b) Delay-DSP element product

Figure 11: The Delay-ALUT product and the Delay-DSP element product.

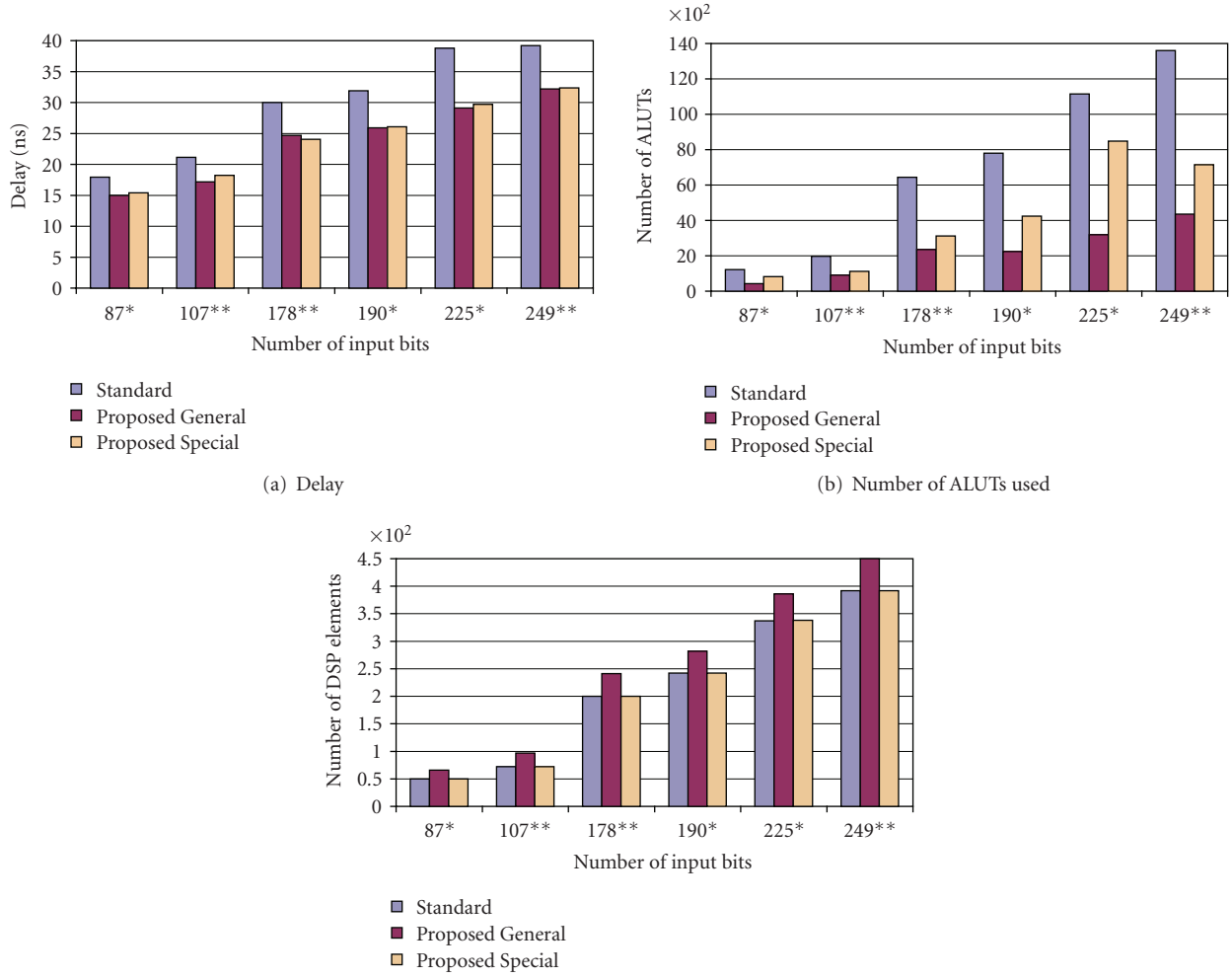


Figure 12: Result comparison for special cases.

required. Based on Altera's FPGAs, the total number of embedded blocks used (in terms of 9-input DSP elements) is equal to $49 \times 8 + (14 \times 2) \times 2 + 1 \times 2 = 450$. However, if there is no second level decomposition, the 18×34-bit multiplication requires a 36×36-bit embedded multiplier. Under this condition, the total number of embedded 9-input DSP elements used is equal to $49 \times 8 + 14 \times 8 + 1 \times 2 = 506$, an increase of 12.5%.

Step 4 (Summing the partial products). The last step is to sum all partial products shown in Figure 8 to get the final result of the large size multiplier. Using the rules proposed in [14], the additions can be performed as shown in Figure 9 based on two-input operand adders.

5. Implementation Results

The multigranular signed multipliers were implemented using Altera's FPGAs. The synthesis tool used is Quartus II version 7.2 targeting the device EP2S180F1508C3 from the Stratix II family [1]. To test the effectiveness of our proposed approach, 9-input DSP element is used as the unit of computation. Results of our proposed design approach are compared with the standard scheme adopted by the Quartus synthesis tool [16], which uses primarily 18×18- and 9×9-bit embedded multipliers as building blocks. All the designs in this paper are registered at the inputs and outputs. The size of input operands is ranged from 40 to 255 bits with an increase of 5 bits from one case to the next one. The case of 256×256-bit multiplier, which has a range of applications, is also assessed. Moreover, some special cases are implemented to reduce the number of embedded blocks based on the algorithms presented in Section 3.3.

The proposed approach and the traditional scheme are compared based on the following metrics extracted from the implementation summary and the timing analyzer summary: (1) the clock period, (2) the number of (Adaptive Look Up Tables) ALUTs used, (3) the number of embedded blocks in terms of 9-input DSP elements used. All these results are presented in Figure 10. Then, the delay-ALUTs product and

Table 3: Result of a 256 × 256-bit multigranular signed multiplier.

	Delay (ns)	Number of ALUTs	Number of DSP elements	Delay-ALUT product	Delay-DSP-element product
Standard	41.530	16050	449	666557	18647
Multigranular	32.487	4577	450	148693	14619
Saving (%)	21.775	71.483	−0.223	77.692	21.600

the delay-DSP-element product are computed based on the implementation results and presented in Figure 11.

Compared to the results of the standard scheme, the proposed multigranular multiplier method has resulted in considerable improvements in terms of timing and area saving. The performance has been improved by 20.7% compared to the standard scheme. For the number of ALUTs used, the multigranular approach consumes an average of 67.6% less area compared to the standard scheme. Although our approach has outperformed the standard method, however, there are roughly 25% cases where our approach requires more number of 9-input DSP elements than that of the standard scheme, as shown in Figure 10(c).

Moreover, the implementation results of the multiplications can be improved using the algorithms of special cases as explained earlier, which allow reducing the number of embedded blocks. These results will be presented later. Also, considering the product of the delay and the number of ALUTs as well as the product of the delay and the number of embedded blocks, a significant improvement has been achieved as it can be noticed from Figure 11. The average reductions of the delay-ALUT product and the delay-DSP element product are 74.4% and 15.8%, respectively, compared to the standard scheme.

For the case of 256 × 256-bit multiplier, results are listed in Table 3. Comparing to the standard scheme, the delay is reduced by about 21.7% and the number of ALUTs saving is up to 71.4%, however, the use of 9-input DSP elements has been increased by one block. This is translated into 0.22% penalty. The delay-ALUT product and the delay- DSP-element product for the multigranular approach have been improved by 77.6% and 21.6%, respectively, compared to the standard scheme.

For the special cases, Figure 12 graphically illustrates the implementation results for six cases ranging from 3 to 7 segments. In this figure, the "proposed general" refers to the design approach presented in Section 3.2. The "proposed special" refers to the special cases, which are implemented using the algorithms presented in Section 3.3. From Figure 12, it is clear that the number of DSP elements used in the special cases is reduced and now it is exactly the same as that used in the standard approach. Although this resulted in an increase in the number of ALUTs, however, it is still significantly less than that used in the standard approach.

6. Conclusions

The focus of this paper is to realize large size signed multipliers using DSP blocks with multigranular embedded signed multipliers in FPGAs. Multiple decompositions are used to efficiently make use of the multigranularity offered in modern FPGAs. The effectiveness of the proposed design approach has been tested using various benchmarks, and compared with a standard approach using commercial tool. Although this tool has complete access to the features available in the DSP blocks and in the 6-LUTs of the target FPGA, however, using our methodology has always outperformed the standard scheme.

References

[1] Altera, "DSP blocks in stratix II & stratix II GX devices," in *Stratix II Device Handbook, Volume 2*, Altera, San Jose, Calif, USA, 2006.

[2] G. Govindu, L. Zhuo, S. Choi, and V. Prasanna, "Analysis of high-performance floating-point arithmetic on FPGAs," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS '04)*, vol. 1, p. 149, Santa Fe, NM, USA, April 2004.

[3] A. Akkas and M. J. Schulte, "A quadruple precision and dual double precision floating-point multiplier," in *Proceedings of the Euromicro Symposium on Digital Systems Design (DSD '03)*, pp. 76–81, Belek-Antalya, Turkey, September 2003.

[4] J. Draper, J. Sondeen, and C. W. Kang, "Implementation of a 256-bit wideword processor for the data-intensive architecture (DIVA) processing-in-memory (PIM) chip," in *Proceedings of the 28th European Solid-State Circuits Conference (ESSCIRC '02)*, pp. 77–80, Florence, Italy, September 2002.

[5] I. Koren, *Computer Arithmetic Algorithms*, A K Peters, Natick, Mass, USA, 2nd edition, 2001.

[6] R. Fried, "Minimizing energy dissipation in high-speed multipliers," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED '97)*, pp. 214–219, Monterey, Calif, USA, August 1997.

[7] C. R. Baugh and B. A. Wooley, "A two's complement parallel array multiplication algorithm," *IEEE Transactions on Computers*, vol. 22, no. 12, pp. 1045–1047, 1973.

[8] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Efficient synthesis of compressor trees on FPGAs," in *Proceedings of the 13th Asia and South Pacific Design Automation Conference (ASP-DAC '08)*, pp. 138–143, Seoul, Korea, January 2008.

[9] B. R. Lee and N. Burgess, "Improved small multiplier based multiplication, squaring and division," in *Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '03)*, pp. 91–97, Napa, Calif, USA, April 2003.

[10] G. Quan, J. P. Davis, S. Devarkal, and D. A. Buell, "High-level synthesis for large bit-width multipliers on FPGAs: a case study," in *Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, vol. 1, pp. 213–218, Jersey City, NJ, USA, September 2005.

[11] N. Nedjah and L. de Macedo Mourelle, "A reconfigurable recursive and efficient hardware for Karatsuba-Ofman's multiplication algorithm," in *Proceedings of IEEE Conference on Control Applications (CCA '03)*, vol. 2, pp. 1076–1081, Istanbul, Turkey, June 2003.

[12] Xilinx Inc., "Xilinx DSP Design Considerations," XtremeDSP for Virtex-4 FPGAs, UG073, (v2.2), 2006.

[13] S. Gao, N. Chabini, D. Al-Khalili, and P. Langlois, "Optimised realisations of large integer multipliers and squarers using embedded block," *IET Computers & Digital Techniques*, vol. 1, no. 1, pp. 9–16, 2007.

[14] S. Gao, D. Al-Khalili, and N. Chabini, "Optimized realization of large-size two's complement multipliers on FPGAs," in *Proceedings of the 50th IEEE International Midwest Symposium on Circuits and Systems Joint with the 5th IEEE Northeast Workshop on Circuits and Systems (NEWCAS '07)*, pp. 494–497, Montreal, Canada, August 2007.

[15] S. Gao, N. Chabini, and D. Al-Khalili, "256×256-bit multiplier using multi-granular embedded DSP blocks in FPGAs," in *Proceedings of the 6th International IEEE Northeast Workshop on Circuits and Systems and TAISA Conference (NEWCAS-TAISA '08)*, pp. 253–256, Montreal, Canada, June 2008.

[16] http://www.altera.com.