

Multipliers for Floating-Point Double Precision and Beyond on FPGAs

Sebastian Banescu, Florent de Dinechin, Bogdan Pasca, Radu Tudoran

▶ **To cite this version:**

Sebastian Banescu, Florent de Dinechin, Bogdan Pasca, Radu Tudoran. Multipliers for Floating-Point Double Precision and Beyond on FPGAs. Highly Efficient Accelerators and Reconfigurable Technologies, Jun 2010, Tsukuba, Japan. ensl-00475781v2

HAL Id: ensl-00475781

https://ens-lyon.hal.science/ensl-00475781v2

Submitted on 1 Nov 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Multipliers for Floating-Point Double Precision and Beyond on FPGAs

LIP Research Report RR2010-15

Sebastian Banescu[2], Florent de Dinechin[1], Bogdan Pasca[1], Radu Tudoran[2]
[1]LIP, projet Arénaire ENS de Lyon
46 allée d'Italie, 69364 Lyon Cedex 07, France
Email: {Florent.de.Dinechin,Hong.Diep.Nguyen,Bogdan.Pasca}@ens-lyon.fr
[2]Computer Science Department
Technical University of Cluj-Napoca, Romania
Email: {Sebastian.Banescu,Radu.Tudoran}@cs.utcluj.ro

***Abstract*—The implementation of high-precision floating-point applications on reconfigurable hardware requires large multipliers. Full multipliers are the core of floating-point multipliers. Truncated multipliers, trading resources for a well-controlled accuracy degradation, are useful building blocks in situations where a full multiplier is not needed.**

This work studies the automated generation of such multipliers using the embedded multipliers and adders present in the DSP blocks of current FPGAs. The optimization of such multipliers is expressed as a tiling problem, where a tile represents a hardware multiplier, and super-tiles represent combinations of several hardware multipliers and adders, making efficient use of the DSP internal resources. This tiling technique is shown to adapt to full or truncated multipliers.

It addresses arbitrary precisions including single, double but also the quadruple precision introduced by the IEEE-754-2008 standard and currently unsupported by processor hardware. An open-source implementation is provided in the FloPoCo project.

***Index Terms*—FPGA, multiplier, truncated multiplier, floating-point, quadruple precision**

I. Introduction

FPGA integration still follows Moore's Law, and FPGAs have been shown to exceed CPU performance in single-precision (or SP, a 32 bit format) and then double-precision (or DP, a 64-bit format including a 52-bit mantissa) [16].

DP arithmetic is popular for commodity and compatibility with software. However, demand for more accuracy is growing, especially in scientific computing [6], and the IEEE-754-2008 revision of the Standard for Floating-Point Arithmetic [10] has introduced a higher precision floating-point format: *quadruple precision* (QP), a 128-bit format including a 112-bit mantissa. So far no general purpose processor offers hardware floating-point units supporting this format. Proprietary core generators such as LogiCore [1] from Xilinx and Megawizard [2] from Altera currently do not scale to QP either.

This article focuses on techniques for building multipliers larger than double precision. There is a special motivation for a QP floating-point multiplier, and one contribution of this work is indeed such a multiplier, however the applications of this work go well beyond that. Multiplication is a pervasive operation, and in an FPGA it should be adapted to its context as soon as this may save resources:

- In many applications, one needs to multiply numbers of different bit-width.
- *Truncated* multipliers [17] discard some of the lower bits of the mantissa to save hardware resources. For a floating-point multiplier, the impact of this truncation can be kept small enough to ensure last-bit accuracy (or faithful rounding) instead of IEEE-754-compliant correct rounding. This small accuracy lost may be compensated by a larger mantissa size. However, it is also perfectly acceptable in situations where a bound on the relative error of the multiplication is enough to ensure the numerical quality of the result. This is for instance the case of polynomial approximation of functions: it is possible to build high-quality functions out of truncated multipliers [4]. In other words, the present work is an important step towards efficient implementations of elementary functions up to quadruple precision on FPGAs.
- The *Karatsuba* technique [3], [5], trading multiplications for additions, can also be used on multipliers, truncated or not.
- *Squarers* are also a special case of multipliers that present optimization opportunities [5].

A contribution of this article is, in Setion III the automation of the tiling technique used manually in [5] – and indeed the automatically-generated multipliers sometimes surpass the hand-crafted ones published there. It is based on a fine modelization of the capabilities of existing DSP blocks. Another contribution is, in Section IV, a novel algorithm for truncated multiplication using embedded multipliers. For QP, the multipliers obtained using this technique save 23 DSP blocks on Virtex4 and 15 DSP blocks on Virtex5.

The operators presented here are freely available as part of the FloPoCo project[1].

II. Background

A. Large multipliers using DSP blocks

Recent FPGAs embed a large number of Digital Signal Processing (DSP) blocks, which include small multipliers. The

[1]www.ens-lyon.fr/LIP/Arenaire/Ware/FloPoCo/

straightforward way of performing large multiplications using these multipliers is to first decompose the large multiplication into a sum of smaller multiplications matching the embedded multipliers. Let α, β be two integer parameters representing the size in bits of each input to an embedded multiplier.

Let A and B be two integers to multiply, of respective sizes $n\alpha$ bits and $m\beta$ bits. The product AB may be written:

$$\begin{aligned} AB &= \sum_{i=0}^{n\alpha-1} a_i 2^i \times \sum_{i=0}^{m\beta-1} b_i 2^i \\ &= \sum_{i,j=0}^{i<n, j<m} 2^{\alpha i+\beta j} A_i B_j \end{aligned}$$

where A_i and B_i are chunks of α and β bits of A and B respectively.

This requires the computation of nm subproducts of size $\alpha \times \beta$, and their summation with the proper weights $2^{\alpha i+\beta j}$. This technique requires nm DSP blocks to implement an $n\alpha + m\beta$ bit multiplier. An automation of this process has been presented in [8] (for $\alpha = \beta$) and in [15] (for $\alpha \neq \beta$ as in Virtex-5/6). Both works focus on the alignment of the subproducts in order to reduce the number of levels of multioperand adder tree. None of these works make use of the internal DSP adders nor address pipelined multipliers. Moreover, as presented in [5], this decomposition process is suboptimal when $\alpha \neq \beta$.

Previous studies [3], [5] have also shown that the Karatsuba technique may reduce the DSP count when $\alpha = \beta$, e.g. from 4 to 3 DSPs when $n = m = 2$, or from 9 to 6 when $n = m = 3$, at the expense of more logic.

B. Relevant DSP features

All DSP blocks contain multipliers. For Xilinx VirtexII-IV and Spartan3 the multiplier size is 18×18 bits signed (or 17×17 bits unsigned). Virtex-5 and Virtex-6 contain rectangular multipliers of 18×25 bits signed (or 17×24 bits unsigned). With respect to section II-A, $\alpha = \beta = 17$ for VirtexII-IV and Spartan3. For Virtex-5/6 the values for the two parameters are $\alpha = 17, \beta = 24$.

In addition to the multiplier, the Xilinx DSP also contains an adder/subtracter unit that can be used to sum two subproducts coming from neighbouring DSPs, possibly with a 17-bit shift. This feature, in combination with four levels of internal registers, may be used to sum up to four shifted subproducts in a pipelined way entirely within four DSP blocks.

The Altera StratixII DSP block contains 4 18×18-bit unsigned multipliers that can also be configured to perform eight 9×9-bit multiplications. Newer generations (StratixIII and IV) allow for an extra configuration performing six 12×12-bit products using the same hardware. A configurable addition tree allows for the four 18×18-bit subproducts to be summed to perform one 36×36-bit multiplication. This adder tree seems to allow a for a similar degree of flexibility as the Xilinx DSP. However, unlike Xilinx', Altera tools currently require Altera-specific primitives to exploit modes where the subproducts do not have equal weights. This requires more development, and for lack of time we therefore focus on Xilinx FPGAs in the rest of this article.

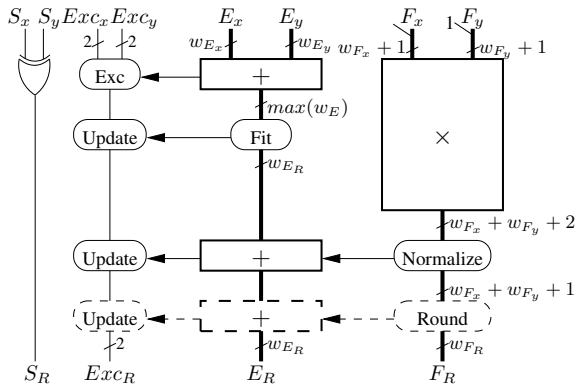


Fig. 1. Architecture of a flexible floating-point multiplier

C. Flexible floating-point multiplication

The floating-point format used in this work is parameterized by exponent size w_E and mantissa fraction size w_F. It is similar in spirit to the IEEE-754 format, but adapted to the context of FPGAs: It does not support subnormals (the possibility of increasing independently the exponent size makes subnormals less relevant in FPGA computing) and encodes exceptions (zero, infinities and Not a Number) in two separate bit to avoid the overhead of coding/decoding them in the exponent field as in the IEEE-754 format.

In addition, we support multiplying numbers of different formats. Let us consider X and Y two floating-point numbers respectively in (w_{E_X}, w_{F_X}) and (w_{E_Y}, w_{F_Y}) formats. The product, noted R, should be on (w_{E_R}, w_{F_R}) format:

$$\begin{aligned} XY &= (-1)^{S_X} 2^{E_X - bias_X} 1.F_X \times (-1)^{S_Y} 2^{E_Y - bias_Y} 1.F_Y \\ &= (-1)^{S_X + S_Y} 2^{E_X - bias_X + E_Y - bias_Y} (1.F_X \times 1.F_Y) \\ R &= (-1)^{S_{XY}} 2^{\diamond_{w_{E_R}}(E_{XY} + bias_R)} \circ_{w_{F_R}} (1.F_R) \end{aligned}$$

The simplified data-path of the fully parametrized floating-point multiplier is presented in Figure 1. There are several differences with respect to the classical version found in textbooks [7], [12] and implemented in most libraries [11], [9], [14] where $w_{E_X} = w_{E_Y} = w_{E_R}$ and $w_{F_X} = w_{F_Y} = w_{F_R}$. Firstly, for $w_{F_X} \neq w_{F_Y}$ the mantissa product requires a rectangular multiplier. Moreover, the result mantissa has to be rounded to w_{F_R} bits ($\circ_{w_{F_R}}$). Secondly, the underflow/overflow conditions change due to the new exponent range. If the exponent result is not representable on w_{E_R} bits than the exception bits have to be respectively updated ($\diamond_{w_{E_R}}$). Finally, the mantissa multiplier will be built using the automated tiling technique which we now present.

III. Tiling

Let us consider our multiplication operands A and B on u and v bits respectively. Our purpose it to multiply A and

B making efficient use of the DSP resources. The technique consists in tiling a $u \times v$ rectangular multiplication board using a minimal number of such multipliers. Starting from the tiled multiplication board, the circuit equation is obtained using a simple rewriting technique.

Tiling, as a reformulation technique for this optimization problem, has been first introduced in [5], where only rectangular tiles were considered. We show in this work that considering more complex tiles allows the tiling technique to optimize the use not only of the multipliers, but also of the adders within DSP blocks.

We take as running example Figure 2(b) (from [5]) in order to introduce tiling for a DP mantissa multiplication on a Virtex5 FPGA. The rectangles denoted by $M1$ to $M8$ are the eight Virtex5 multiplier tiles used to perform the multiplication (17×24 bits). The central 10×10-bit multiplication might be either performed in logic if the DSP count is a big constrain, either partially using one DSP block.

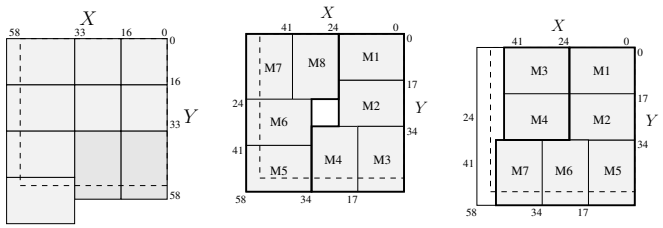
Each rectangle represents the product between a range of bits of X and Y. For example $M1 = X_{0:23} \times Y_{0:16}$. For each rectangle, the ranges of X and Y correspond to its projection on the X and Y axis respectively. A rectangle has a weighted contribution to the final product, the weight being equal to the sum of its upper right corner coordinates (e.g. the weight of the $M4$ tile is 2^{17+34}). The presented rewriting technique yields:

$$\begin{array}{lll} XY & = & \left(M1 + 2^{17}M2 + 2^{34}M3 + 2^{51}M3\right) \qquad S_0 \\ & +2^{24} & \left(M8 + 2^{17}M7 + 2^{34}M6 + 2^{51}M5\right) \qquad S_1 \\ & +2^{48} & M_{Logic} \end{array}$$

We have parenthesized the equation in order to make full use of the Virtex5 internal DSP adders (see section II-B). Due to the fixed 17-bit shifts between the operands, each sub-sum S_0 and S_1 may be computed entirely using DSP block resources. This reduces the number of inputs of the final multi-operand adder to three.

Such a parenthesing involving only 17-bit shifts is graphically descried as a *super-tile*. Figure 3 shows some super-tiles corresponding to the DSP capabilities of Virtex 4 and 5/6. These super-tiles (and all their subsets) don't require additional hardware to perform the full product. In addition, larger super-tiles can be obtained by coupling the black and white circles of adjacent super-tiles. This corresponds to using the cascading adder input of the DSP blocks. Actually, all the possible super-tiles may be generated by the primitives shown on Figure 4.

On Stratix, the large adders inside the DSP block that can be used to add up to four 18x18-bit partial products having the same magnitude. This corresponds to a line of tiles parallel to the main diagonal. However, as previously stated, we are currently unable to obtain the predicted performance out of the Altera Quartus tools. This could be solved by using Alter-specific primitives, but would require much more development work.



(a) Xilinx (b) Tiling in [5] (c) Proposed

Fig. 2. 53-bit multiplication using Virtex-5 DSP48E. The dashed square is the 53x53 multiplication.

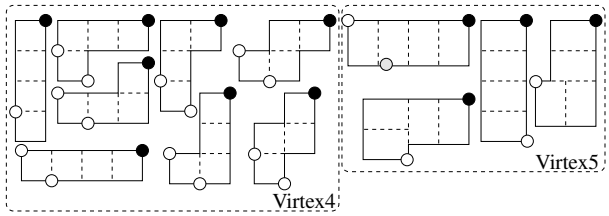


Fig. 3. Some super-tiles exactly matching DSP blocks

A. Design Decisions

In the previous example, there remains an untiled 10-bit $\times$ 10-bit square. Should this be implemented as logic, or as an underutilized DSP block? This is a trade-off between logic and DSP blocks, and as such the decision should be left to the user. This situation is very common, for instance there is also an untiled part in Figure 2(c). We have therefore decided to offer the user the possibility to select a ratio between DSP count and logic consumption. This *ratio* is as a number in the $[0, 1]$ range. Larger values for the ratio favour DSP oriented architecture whereas lower values favour logic oriented architectures. The total number of multipliers used is a function of the input widths, ratio and FPGA target.

In order to exploit this user-provided ratio accurately, we have modelled the logical equivalence of a DSP block for various FPGA families, inside FloPoCo's `Target` hierarchy.

B. Algorithm

The construction of a tentative multiplier configuration consists of three steps.

1) Generate a valid partition of the large multiplication into smaller partial products or tiles.
2) Group these tiles as super-tiles in order to reduce the number of operands of the large multiplier's final adder. The super-tiles are built using the regrouping primitives presented in Figure 4. Two successive tiles can be regrouped if their their black and white circles correspond to one of the regrouping primitives. When building super-tiles we also balance their sizes in order to reduce operator pipeline depth and the number of synchronization registers.
3) Compute the approximate cost of the configuration. This cost includes: the DSPs, the slices needed for

computing the rest of the multiplication, and the cost of the multioperand adder used to compute the final result.

Configurations may be compared according to this cost. The best one will be chosen, and its VHDL generated.

Choosing among *all* possible configurations takes an exponential number of steps with respect to the size of the multiplication board $O((u \times v)^\delta)$, where u and v are the dimensions of the multiplication and δ is the number of DSPs. Although this would ensure we find the optimal configuration, the exponential complexity prevents from obtaining results in reasonable time. Hence, we prune exploration branches using the following criteria:

- Tiles do not overlap. In step 1, we only consider tilings which align tile edges. This reduces the number of tilings to O(2^δ) for Virtex4 and O(3^δ) for Virtex5.
- Configurations symmetrical to already existing ones are pruned.
- Configurations where large holes appear inside the tiling are also pruned.

C. Reality check

We have used the presented algorithm in order to generate mantissa multipliers for DP (53bit) and QP (113bit) floating-point. Table I presents the synthesis results obtained for both the mantissa multiplier and the complete floating-point multiplier, on Virtex4 (xc4vfx100-12-ff1152) and Virtex5 (xc5vfx100T-3-ff1738) FPGAs using Xilinx ISE 11.4. The results of this work are compared to Xilinx Logicore core generator, a double precision operator presented in [5] and combinatorial results obtained from [15]. With respect to the results presented in [5] we manage to offer an DP mantissa multiplier operator that saves 2 DSP blocks at the expense of some logic while running at a similarly high frequency. With respect to [15] we offer high performance operators while reducing the number of DSP blocks. The biggest difference is for DP, where their decomposition technique infers 12 DSPs, out of which several are underutilized. With respect to Xilinx Logicore, we manage to save DSP blocks without big penalties in logic consumption. For example, for Virtex4 we are able to save 6 DSPs for approximately 330 slices.

IV. Tiling Truncated multipliers

Truncated multipliers reduce resources, delay, or power consumption [17], [13]. Let us consider two integers A and B on u and v bits respectively with AB on $n = u + v$ bits. The idea is to save the computation of some of the less significant columns in the multiplication array (see the greyed-out rows in Figure 5(a)) so that the error of the integer multiplication remains small enough. More precisely, given a target precision

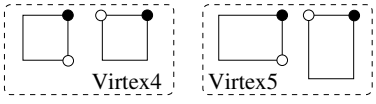
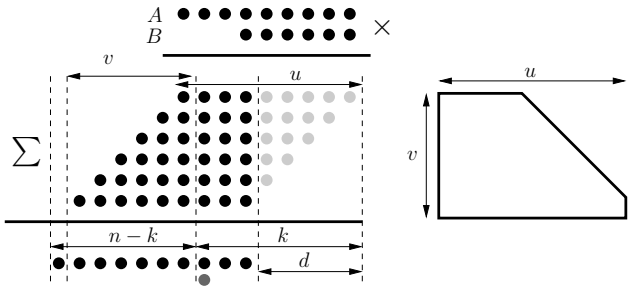


Fig. 4. Super-tiling primitives



(a) truncated multiplication (b) truncated board

Fig. 5. Truncated multiplication and the corresponding tiling multiplication board

weight k, we build a multiplier that returns a result faithfully rounded on $n-k$ bits. Faithful rounding means that the total error is smaller that the weight of the last bit of the result: $E_{total} \leq 2^k$.

A. Faithfully accurate multipliers

Let us first determine the maximum number of columns, denoted by d, that may be removed (see Figure 5(a)).

The error E_{total} has two components, $E_{total} = E_{approx} + E_{round}$, where E_{approx} is the approximation error introduced by the truncation of the d columns, and E_{round} is the error of rounding the $n-d$-bit intermediate result to $n-k$ bits.

To ensure that $E_{total} \leq 2^k$, we need to distribute our 2^k error budget between the two error sources. By adding a single one to the multiplier array (the grey dot on Figure 5(a)) before summing it to an $n-d$-bit number, the truncation of this number to $n-k$ bits implements round to nearest, thus ensuring $E_{round} \leq 2^{k-1}$. The remaining 2^{k-1} are allocated to E_{approx}.

The sum of the first d discarded columns is in the interval $0 \leq E_{approx} \leq \sum_{i=1}^{d} i2^{i-1} = (d-1)2^d + 1$ (see Figure 5(a)). An offset correction bit can reduce this error by almost half by centering it [17]. Combined with the previous constraint $E_{approx} < 2^{k-1}$, this provides us a relation of the form $d = f(k)$. Table II shows how the number of discarded columns varies for common floating point formats.

TABLE II
Truncated multipliers providing faithful rounding for common floating point formats

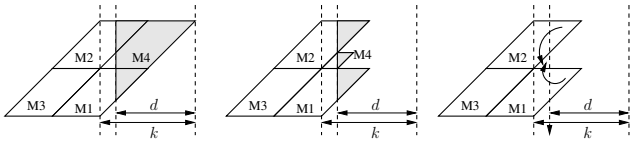
Precision	k	Discarded (d)
Single	23	18
Double	52	46
Quadruple	112	105

B. FPGA Fitting

The theoretical saves in complexity entailed by truncated multiplications approaches 50%. The entailed saves have two components: the size of the computed subproducts and the size of the operands in the multioperand reduction scheme. The truncation technique applied to a multiplication performed using DSP blocks is presented in Figure 6(a). The architecture

TABLE I
COMPARISON OF MULTIPLIER IMPLEMENTATIONS

(w_E, w_F)	Tool, FPGA, Freq.	Mantissa multiplier $(w_F+1)\times(w_F+1)$	Complete floating-point multiplier
(11,52)	ours, Virtex4, 400MHz	11cycles @ 368MHz, 595sl., 10DSP	16cycles @ 338MHz, 729sl. 10DSP
(15,112)	ours, Virtex4, 400MHz	18cycles @ 358MHz, 1741sl., 49DSP	25cycles @ 319MHz, 2125sl., 49DSP
(15,112)	Virtex4,[15]	0cycles @ 76MHz, 1100sl., 49DSP	
(11,52)	ours, Virtex5, 400MHz	9cycles @ 407MHz, 530LUT 506REG 9DSP	14cycles @ 407MHz, 804LUT 804REG 9DSP
(11,52)	ours, Virtex5, 400MHz	8cycles @ 407MHz, 919LUT 872REG 6DSP	13cycles @ 407MHz, 1184LUT 1080REG 9DSP
(11,52)	Virtex5, [5] Fig.2(b)	4cycles @ 369MHz, 243LUT 400REG 8DSP	
(11,52)	Virtex5,[15]	0cycles @ 111MHz, 200LUT 12DSP	
(15,112)	ours Virtex5, 400MHz	13cycles @ 407MHz, 2070LUT 2062REG 34DSP	20cycles @ 355MHz, 2978LUT 2815REG 34DSP
(15,112)	Virtex5,[15]	0cycles @ 90MHz, 1000LUT 35DSP	
(11,52)	Logicore, Virtex4	18cycles @ 400MHz, 279sl., 16DSP	22cycles @ 321MHz, 561sl. 16DSP
(11,52)	Logicore, Virtex5 Fig.2(a)	12cycles @ 450MHz, 229LUT 280REG 10DSP	18cycles @ 319MHz, 339LUT 482REG 10DSP



(a) wasteful (b) better (c) compensated

Fig. 6. Truncation applied to multipliers. Left: Classical truncation technique applied to DSPs. Center: Improved truncation technique. M4 is computed using logic. Right: FPGA optimized compensation technique. M4 is not computed.

consumes 4 DSPs to compute the subproducts M1-M4. The greyed out parts of these subproducts are then discarded before performing the final addition. Out of the 4 DSPs used, 2 are softly underutilized (M1 and M2) and one is greatly underutilized (M4). A better architecture that performs M4 in logic is presented in figure 6(b). This architecture saves one DSP block at the expense of the logic used to perform M4, which can be itself truncated.

However, on both Figure 6(a) and 6(b), the monolithic DSP blocks compute all the bits of M1 and M2. As these bits come for free, we may take them into account, as it will reduce E_{Approx} and possibly allow us to increase d. This requires adders extending beyond $n-d$, but those are for free if they are inside the DSP blocks.

We therefore want to tile the truncated multiplier such that the error entailed by discarding the untiled part meets the previously defined error budget. In this way, the bits not computed at the left of k will be compensated by the ones computed at the right, as illustrated on Figure 6(c).

C. Architecture generation algorithm

A two phase algorithm was implemented in order to generate truncated multiplier using the previously presented tiling technique. The first phase tiles the multiplication board starting from bottom left using $\delta = \lfloor Area_{board}/Area_{tile} \rfloor$ DSPs where $Area_{board}$ is the area of a multiplication board similar in shape to that in Figure 5(b) (size is dependent on k) and $A_{tile} = \alpha \times \beta$. By construction, the approximation error of this tiling, E_{Approx}, will be larger than 2^{k-1}.

The second phase reduces E_{Approx} so that it becomes smaller than 2^{k-1}. In order to do this, we rely on pipelined soft-core multipliers (pipelined multipliers using logic-only). E_{Approx} can be reduced by tiling some high-weighted yet untiled bits. Taking Figure 7 as running example, these are the untiled bits situated further away (Euclidean distance) from the origin (top right corner).

The second phase of the algorithm finds at each step the furthest point from the origin. If this point is adjacent to an already existing soft-core multiplier , it increases the respective dimension of this multiplier. Otherwise, an 1×1 bit soft-core multiplier is instantiated at that point. If the soft-core multiplier size is equal to that of a DSP block, it is replaced by such a block. Next, the error produced by the new configuration is evaluated. The second phase iterates until the 2^{k-1} approximation error budget is met. Figure 7 shows how the size these soft-core multipliers increases. When a valid configuration is met, its hardware cost is evaluated, and stored if minimal. If possible, a new tiling is explored and cost is re-evaluated.

We remark that with respect to the classical truncation algorithm, not all the bits at the left of the virtual truncation line are computed. In fact, the bits computed for free at the right of this line compensate them. The extra cost of this architecture comes from the few extra bits of the operands in the final multi-operand addition.

Figure 8 shows some possible tilings for large precision truncated multipliers. Table III presents synthesis results for DP and QP. Using our improved truncate multiplier technique we are able to reduce significantly reduce the number of DSPs with respect to classical multiplications. For example, on Virtex4 for DP we are able to reduce DSP count from 10 to

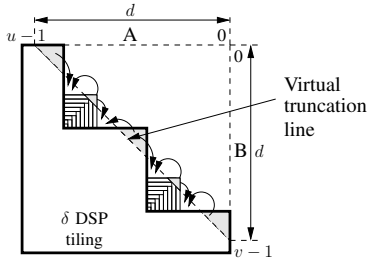
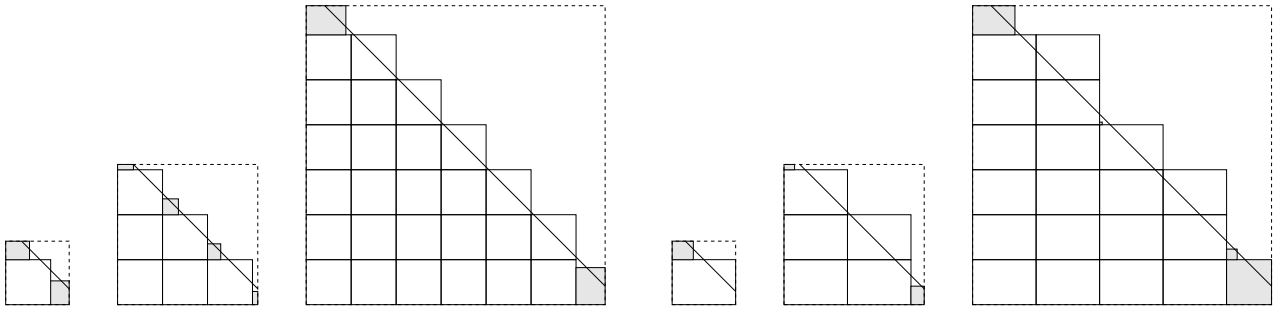


Fig. 7. Tiling truncated multiplier using DSPs and soft-core multipliers



(a) SP (b) DP (c) QP (d) SP (e) DP (f) QP

Fig. 8. Mantissa multipliers for SP,DP,QP, Virtex4 (left) and Virtex5 (right) ensuring faithful rounding. The grey tiles represent soft-core multipliers

TABLE III
TRUNCATED MULTIPLIER RESULTS

FPGA	Prec.	Latency, Freq.	Resources
Virtex5	DP	6 cycles @ 414MHz	320LUT 302REG 5DSP
	QP	20 cycles @ 334MHz	2497LUT 2321REG 19DSP
	QP	14 cycles @ 245MHz	2249LUT 1576REG 19DSP
Virtex4	DP	11 cycles @ 368MHz	358sl. 7DSP
	QP	21 cycles @ 368MHz	1735sl. 26DSP

7 DSPs while also reducing slice count and for QP we reduce from 49 to 26 at without any slice penalty. On Virtex5, the reductions are from 6 to 5 for and roughly half the LUTs and REGs for DP and from 34 to 19 at a small increase in logic resources.

V. CONCLUSION

This article addresses the construction large precision multipliers working at high frequencies, from specifications including operand size, deployment target, running frequency, and optimization directives.

By automating the tiling technique presented in [5], we are able to offer a fully parametrized multiplier operator generator which is capable of generating operators that sometime surpass the hand-crafted ones.

We have also extended this technique to the generation of faithful truncated multipliers, and applied it to build faithfully rounded floating-point multipliers. The savings entailed by this approach are significant, and this type of multiplier could be preferred when IEEE-754 compliance is not mandatory. Moreover, these multipliers can be applied to the polynomial evaluation used to build high-quality functions for FPGAs [4], where only an error bound is required for the final result.

Future work includes finalizing an Altera version for both regular and truncated tiling multipliers, and extending tiling-based approaches to squarers and Karatsuba multipliers.

REFERENCES

[1] ISE 11.4 CORE Generator IP.

[2] MegaWizard Plug-In Manager.

[3] J.-L. Beuchat and A. Tisserand. Small multiplier-based multiplication and division operators for Virtex-II devices. In *Field-Programmable Logic and Applications*, 2002.

[4] F. de Dinechin, M. Joldes, and B. Pasca. Automatic generation of polynomial-based hardware architectures for function evaluation. In *Application-specific Systems, Architectures and Processors*. IEEE, 2010.

[5] F. de Dinechin and B. Pasca. Large multipliers with fewer DSP blocks. In *Field Programmable Logic and Applications*. IEEE, Aug. 2009.

[6] F. de Dinechin and G. Villard. High precision numerical accuracy in physics research. *Nuclear Inst. and Methods in Physics Research, A*, 559:207–210, 2006.

[7] M. D. Ercegovac and T. Lang. *Digital Arithmetic*. Morgan Kaufmann Publishers, 2004.

[8] S. Gao, N. Chabini, D. Al-Khalili, and P. Langlois. Optimised realisations of large integer multipliers and squarers using embedded blocks. *IET Computers & Digital Techniques*, 1(1):9–16, 2007.

[9] G. Govindu, L. Zhuo, S. Choi, and V. Prasanna. Analysis of high-performance floating-point arithmetic on FPGAs. In *Reconfigurable Architecture Workshop*, 2004.

[10] IEEE Computer Society. *IEEE Standard for Floating-Point Arithmetic, IEEE Std 754-2008*. 2008.

[11] J. Liang, R. Tessier, and O. Mencer. Floating point unit generation and evaluation for FPGAs. *Field-Programmable Custom Computing Machines*, page 185, 2003.

[12] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres. *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2010.

[13] M. J. Schulte, K. E. Wires, and J. E. Stine. Variable-Correction Truncated Floating Point Multipliers. In *Asilomar Conference on Signals, Circuits and Systems*, pages 1344–1348, 2000.

[14] R. Scrofano, G. Govindu, and V. K. Prasanna. A Library of Parameterizable Floating-Point Cores for FPGAs and Their Application to Scientific Computing. In *Engineering of Reconfigurable Systems and Algorithms*, pages 137–148. CSREA Press, 2005.

[15] S. Srinath and K. Compton. Automatic generation of high-performance multipliers for FPGAs with asymmetric multiplier blocks. In *Field Programmable Gate Arrays*, pages 51–58, New York, NY, USA, 2010. ACM.

[16] K. Underwood. FPGAs vs. CPUs: trends in peak floating-point performance. In *Field Programmable Gate Arrays*, pages 171–180. ACM, 2004.

[17] K. E. Wires, M. J. Schulte, and D. McCarley. FPGA Resource Reduction Through Truncated Multiplication. In *Field-Programmable Logic and Applications*, pages 574–583. Springer-Verlag, 2001.