Multipliers for Floating-Point Double Precision and Beyond on FPGAs

Article *in* ACM SIGARCH Computer Architecture News · January 2011

DOI: 10.1145/1926367.1926380 · Source: OAI

4 authors:

Sebastian Banescu
Quantstamp
57 PUBLICATIONS **827** CITATIONS
SEE PROFILE

Florent De Dinechin
Institut National des Sciences Appliquées de Lyon
164 PUBLICATIONS **3,444** CITATIONS
SEE PROFILE

Bogdan Pasca
Intel
65 PUBLICATIONS **1,312** CITATIONS
SEE PROFILE

Radu Tudoran
Institute for Research in IT and Random Systems
44 PUBLICATIONS **602** CITATIONS
SEE PROFILE

Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON-UCBL n° 5668

Multipliers for Floating-Point Double Precision and Beyond on FPGAs

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

* Technical University of Cluj-Napoca, Romania
Sebastian.Banescu@cs.utcluj.ro, Radu.Tudoran@cs.utcluj.ro

** LIP, Arénaire
CNRS/ENSL/INRIA/UCBL/Université de Lyon
46, allée d'Italie, 69364 Lyon Cedex 07, France
Florent.de.Dinechin@ens-lyon.fr,Bogdan.Pasca@ens-lyon.fr

April 2010

Research Report N° 2010-15

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : `lip@ens-lyon.fr`

Multipliers for Floating-Point Double Precision and Beyond on FPGAs

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

* Technical University of Cluj-Napoca, Romania

Sebastian.Banescu@cs.utcluj.ro, Radu.Tudoran@cs.utcluj.ro

** LIP, Arénaire

CNRS/ENSL/INRIA/UCBL/Université de Lyon

46, allée d'Italie, 69364 Lyon Cedex 07, France

Florent.de.Dinechin@ens-lyon.fr,Bogdan.Pasca@ens-lyon.fr

April 2010

Abstract

The implementation of high-precision floating-point applications on reconfigurable hardware requires a variety of large multipliers: Standard 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 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 are the wiring of several hardware multipliers 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 in 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.

Keywords: multiplier, truncated multiplier, floating-point, quadruple precision

1 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) [17].

IEEE-754 compliant DP arithmetic is popular for commodity and compatibility with software. However, demand for more accuracy is growing, especially in scientific computing [6], and the new revision of the IEEE-754 Standard for Floating-Point Arithmetic [3] 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 from Xilinx and Altera, *LogiCore* [1] and *Megawizard* [2] 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. We have identified several such situations that are relevant to the building of floating-point operators.

- In many applications, one needs to multiply numbers of different bit-width.
- *Truncated* multipliers [19] 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 [18]. In other words, the present work is an important step towards efficient implementations of elementary functions up to quadruple precision on FPGAs.
- The Karatsuba-Offman technique, that trades multiplications for additions, can also be used on multipliers, truncated or not.

The technique itself is 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.

A novel algorithm for truncated multiplication using embedded multipliers is given presented in Section 3.4.2. The multipliers obtained using this technique save 23DSP blocks on Virtex4 and 15DSPs on Virtex5 for QP.

2 Background

2.1 Large multipliers using DSP blocks

Recent FPGAs embed a large number of Digital Signal Processing (DSP) blocks, which include small multipliers.The 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 k_1, k_2 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 nk_1 bits and mk_2 bits. The product AB may be written:

$$\begin{aligned} AB &= \sum_{i=0}^{nk_1-1} a_i 2^i \times \sum_{i=0}^{mk_2-1} b_i 2^i \\ &= \sum_{i,j=0}^{i<n, j<m} 2^{k_1 i + k_2 j} A_i B_j \end{aligned}$$

where A_i and B_i are chunks of k_1 and k_2 bits of A and B respectively.

This requires the computation of nm subproducts of size $k_1 \times k_2$, and their summation with the proper weights $2^{k_1 i + k_2 j}$. This technique requires nm DSP blocks to implement an $nk_1 + mk_2$ bit multiplier. An automation of this process for square multipliers has been presented in [9] and for the rectangular multipliers of Virtex-5/6 in [16]. Both works focus on the alignment of the subproducts in order to reduce the number of levels of multioperand adder tree. Unfortunately none of these works make use of the internal DSP adders nor they concern pipelined multipliers. Moreover, as presented in [5] this decomposition process is suboptimal for rectangular multipliers.

Previous studies [4, 5] also have shown that the Karatsuba-Offman technique may reduce the DSP count when $k_1 = k_2$, 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.

2.2 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 2.1, $k_1 = k_2 = 17$ for VirtexII-IV and Spartan3. For Virtex-5/6 the values for the two parameters are $k_1 = 17, k_2 = 24$.

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.

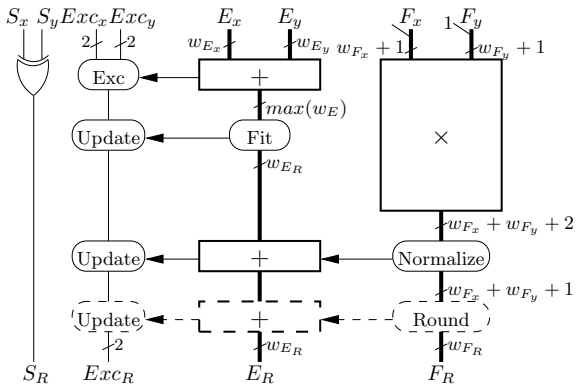


Figure 1: The architecture of a flexible floating-point multiplier

2.3 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.

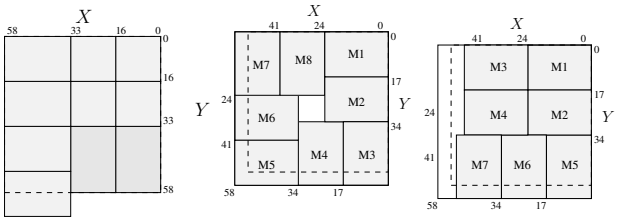
In addition, we support multiplying numbers of different formats. Let us consider X and Y two floating-point numbers respectively in (w_{E_X}, w_{F_X}) and (w_{E_Y}, w_{F_Y}) formats. The product, noted R, should be on $(w_{E_R} + w_{F_R})$ format:

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

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 [8, 13] and implemented in most libraries [12, 10, 15] 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 presented in Section 3.

3 Tiling

Let us consider our multiplication operands A and B on p and q bits respectively. Our purpose it to multiply A and B making efficient use of the DSP resources. The technique consists in



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

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

tiling a $p \times q$ rectangular board using a minimal number of such multipliers. Starting from the tilled 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]) 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:

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

We have parenthesized the equation in order to make full use of the Virtex5 internal DSP adders (see section 2.2). 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.

In the following sections we will detail on the design decisions and algorithms chosen to automate this process.

3.1 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

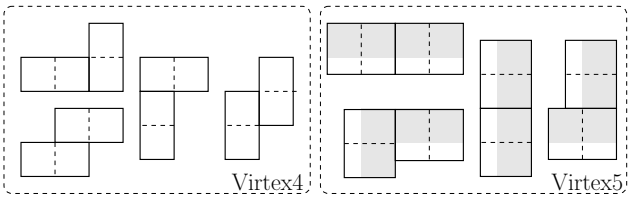


Figure 3: Coarser tiles making efficient use of DSP internal adders. On the left are the layouts for both Virtex4 and Virtex5, on the right the specific layouts for Virtex4

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.

The tiles represent the hardware multipliers inside the DSP blocks. In order to reach the best configuration we need not take into consideration only multiplications, but also additions and shifts that can be done inside a DSP block. Hence after reaching a certain configuration, it is used as input for another algorithm that finds the optimal way to associate tiles into *supertiles* such that the maximum amount of resources inside the DSP block are used.

3.2 Algorithm

The proposed algorithm is iterative and runs in three steps: The first step is generating a valid partition of the large multiplication into smaller partial products. The partial products are computed either by hardware multipliers inside DSP blocks or using logic slices.

The second step is finding the optimal binding between partial products in order to reduce the number of operands that input the final adder which produces the large multiplication result. In the case of Virtex boards, when placing the DSPs on the board we must not consider only the covered space, but to consider whether or not we can perform a shift between DSPs. If we can do this we can save a level in the final adder that will compute the final result. The valid directions of shifts are the one presented in Figure 3. These direction kernels, possibly chained, represent the kernels used to build supertiles. At this step we also balance the sizes of the supertiles in order to reduce operator pipeline depth and therefore shift-registers.

On Stratix boards shifting cannot be performed as for Virtex targets in the cases of 9, 12 and 18-bit multiplier modes. Nevertheless we have large adders inside the DSP block that can be used to add up to 4 partial products having the same magnitude. Multipliers that can be associated in this way appear to be positioned collinearity, parallel to the main diagonal.

The third step is computing the approximate cost of the configuration after the superblocks at step 2 were created. The cost of the configuration 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. If the current configuration has the lowest cost found yet, it is recorded and its cost is the new minimum.

After comparing configurations the best one found is chosen for implementation.

Choosing among *all* possible configurations takes an exponential number of steps with

Table 1: Tiling multiplier results

wE, wF	**Tool, FPGA, Freq.**	**Mantissa Multiplier** $2(w_F+1)$	**FPMultiplier**
(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,[16]	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]	4cycles @ 369MHz, 243LUT 400REG 8DSP	
(11,52)	Virtex5,[16]	0cycles @ 111MHz, 200LUT 12DSP	
(15,112)	ours Virtex5, 400MHz	13cycles @ 407MHz, 2070LUT 2062REG 34DSP	20cycles @ 355MHz, 2978LUT 2815REG 34DSP
(15,112)	Virtex5,[16]	0cycles @ 90MHz, 1000LUT 35DSP	
(11,52)	Logicore, Virtex4	18cycles @ 400MHz, 279sl., 16DSP	22cycles @ 321MHz, 561sl. 16DSP
(11,52)	Logicore, Virtex5	12cycles @ 450MHz, 229LUT 280REG 10DSP	18cycles @ 319MHz, 339LUT 482REG 10DSP

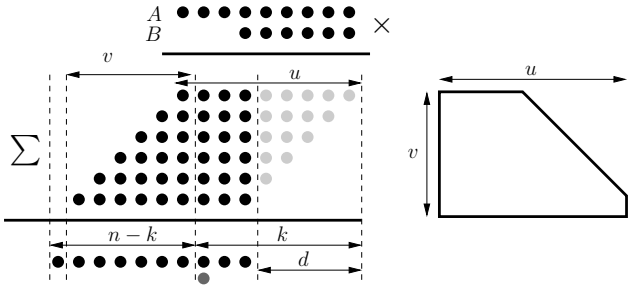
respect to the size of the board $O((w \times h)^n)$, where w and h are the dimensions of the multiplication and n is the number of DSPs. Although this would ensure the optimal configuration, the exponential complexity prevents from obtaining results in reasonable time. Hence, we prune exploration branches using the following criteria:

- DSPs do not overlap. The placement step will try 6 positions placed in the upper-right corner for the first DSP. The rest of the DSPs will be positioned only in positions that maximize the potential of creating good solutions. Hence, for the second DSP only 2 or 3 positions will be tried and so one. The number of steps is reduced to O(2^n) for Virtex4 or O(3^n) Virtex5.

- branches in which DSP indexes are swapped are similar. A number is assigned to each DSP. We prune whenever a higher order DSP would be positioned in the right or above a lower order DSP. Moreover, we also prune when the remaining space on the board is insufficient to place the remaining number of DSPs.

3.3 Reality check

We have used the presented algorithm in order to perform mantissa multiplications for required for DP(53bit) and QP(113bit) floating-point multiplication. Table 3.3 presents the synthesis results obtained for both mantissa multipliers and the full floating-point multiplier operator for Virtex4 (xc4vfx100-12-ff1152) and Virtex5 (xc5vfx100T-3-ff1738) FPGAs using Xilinx ISE 11.4. The results of this work are compared to the *state of the art* Xilinx Logicore core generator, a double precision operator presented in [5] and combinatorial results obtained from [16]. With respect to the results presented in [5] we manage to offer an DP mantissa multiplier operator that saves 2DSP blocks at the expense of some logic while running at a similarly high frequency. Whit respect to [16] we offer high performance operators while reducing the number of DSP blocks. The biggest difference is for DP where their decomposition technique infers 12DSPs 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 6DSPs for approximately 330slices.



(a) truncated multiplication (b) truncated board

Figure 4: Truncated multiplication and the corresponding tiling board

3.4 Tiling Truncated multipliers

Truncated multipliers are used in order to reduce FPGA resources, delay or power consumption [19, 14]. Given a target precision p, the idea idea is to remove some of the less significant multiplication matrix columns(see the greyed-out rows in Figure 4(a)) so that the approximation error remains smaller than p. This reduces resource usage while offering the required precision at the output.

Let us consider two integers A and B on u and v bits respectively with AB on $n = u + v$ bits. The target precision weight is denoted by k such that $p = 2^k$. Our objective is to determine the maximum number of columns to be removed (denoted by d) such that our error hypothesis holds.

The error introduced by this technique, E_{total}, has two components. First, E_{approx} is the approximation error introduced by the truncation of the d columns. Second, the approximated result has to be rounded to $n - k$ bits which entails a rounding error.

$$E_{total} = E_{approx} + E_{round}$$

In order to obtain faithful rounding on $n - k$ bits we need to ensure that $E_{total} \leq 2^k$. Therefore, we need to distribute our 2^k error budget between our two error sources. By choosing *round to nearest* rounding we ensure that $E_{round} \leq 2^{k-1}$. The remaining 2^{k-1} are allocated to E_{approx}.

The sum of the first d discarded columns is in the interval $[(d-1)2^d+1, 0]$. The truncation error is of the order $t = \lceil \log_2((d-1)2^d + 2) \rceil$. An offset correction term $c = 2^{t-1}$ is used to reduce the truncation error order to $t - 1$. Moreover, $E_{approx} \leq 2^{k-1}$ and therefore $t \leq k$ which gives us a relation of the form $d = f(k)$. Table 2 shows how the number of discarded columns varies for common floating point formats.

Table 2: The number of truncated multiplication matrix columns for common floating point formats

Precision	k	Discarded (d)
Single	23	18
Double	52	46
Quadruple	111	105

3.4.1 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 presetned in Figure 6(a). The architecture 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 replaces 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. Nevertheless, we can improve on this.

3.4.2 New Architecture

We start from the observation that holds for both Figure 6(a) and 6(b): in order to ensure that all bits left of the discarded d columns are computed we need to wastefully compute some bits that are later discarded (Figure 6(a),6(b), the greyed-out bits). However, we can use the information from these bits in order to save hardware.

We will tile the board with multipliers such that the error entailed by discarding the untiled part meets previously defined the error budget. In this way, the bits not computed at the left of k will be compensated by the ones computed at the right Figure 6(c).

A two phase algorithm was implemented in order to generate truncated multiplier using the previously presented tiling technique. The first phase tiles the board starting from bottom left using the $\delta = \lfloor Area_{board}/Area_{tile} \rfloor$ DSPs where $Area_{board}$ is the area of a board similar to that in Figure 4(b) (size is dependent on k) and $A_{tile} = DSP_{width} \times DSP_{height}$. By construction, the compensated approximation error of this tiling, $E_{Approx}^{compensated}$, will be larger than 2^{k-1}.

The second phase is used in order to reduce $E_{Approx}^{compensated}$ 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). $E_{Approx}^{compensated}$ can be reduced by tiling some high-weighted yet untiled bits. Taking Figure 5 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 point the furthest point from the origin. If this point is adjacent to an already existing soft-core multiplier in that point, it increases the size of this multiplier. Otherwise, an 1×1 bit soft-core multiplier is instantiated at that

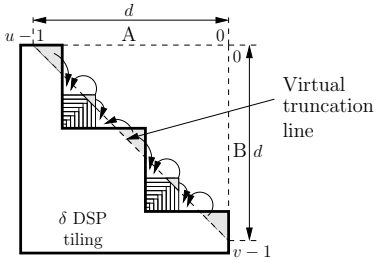
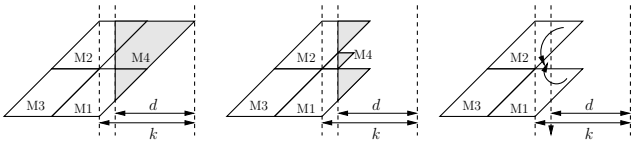


Figure 5: Tiling truncated multiplier using DSPs and soft-core multipliers



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

Figure 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

point. If the soft-core multiplier size is equal to that of a DSP block, it is replaced by this one. Next, the cost of the error produced by the new configuration is evaluated. The second-phase runs until the 2^{k-1} approximation error budget is met. Figure 5 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 at the right of this line compensate for these bits. The extra cost of this architecture comes from the few extra bits of the operands in the final multi-operand addition.

Figure 7 shows some possible tilings for large precision truncated multipliers. Table 3 presents synthesis results for DP and QP. We using the novel truncate multiplier technique we are able to reduce significantly reduce the number of DSPs. For example, on Virtex4 for DP we are able to reduce it from 10 to 7 DSPs while also reducing slice count and for QP we reduce from 49 to 26 at the expense of approximately 1500 slices. 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 LUT count.

4 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.

In the context of multipliers we have managed to automate the tiling technique presented in [5]. We have offered a fully parametrized multiplier operator generator which is capable of generating operators that sometime surpass the hand-crafted ones.

We have also presented a novel approach to the technique of truncated multipliers which adapts well to multipliers build using DSP blocks. The technique was applied to building mantissa multipliers for faithfully rounded floating-point multipliers. The saves entailed by the technique are significant and this type of multiplier could be preferred when IEEE-754

Table 3: 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

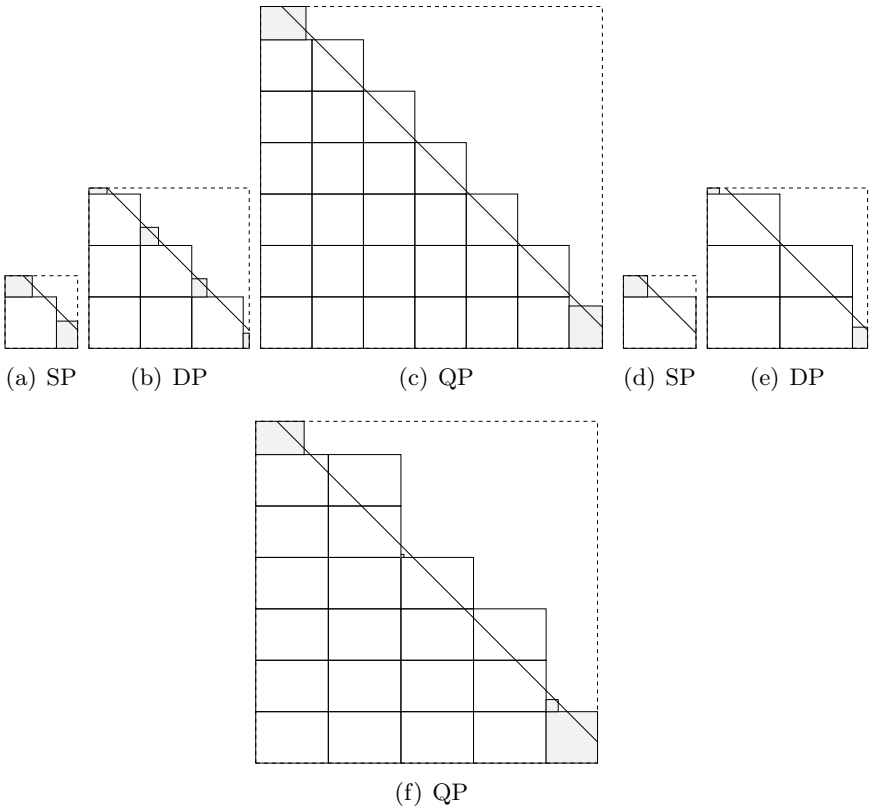


Figure 7: Mantissa Multipliers for SP,DP,QP, Virtex4 (left) and Virtex5(right) ensuring faithful rounding. The grey tiles represent soft-core multipliers

compliance is not mandatory. Moreover, these multipliers can be applied to the polynomial evaluation used to build high-quality functions for FPGAs [18] where only an error bound is required for the final result.

Future work includes finalizing an Altera version for both regular tiling and truncated tiling and extending the technique to squarers. Moreover, we plan to introduce Karatsuba-Offman supertiles which will reduce DSP count for QP mantissa multiplication from 49 to 34 on Virtex4.

References

[1] ISE 11.4 CORE Generator IP.

[2] MegaWizard Plug-In Manager.

[3] IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008*, pages 1 –58, 29 2008.

[4] J. Beuchat and A. Tisserand. Small multiplier-based multiplication and division operators for Virtex-II devices, 2002.

[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] J. Detrey and F. de Dinechin. FPLibrary: operators for the design of "real number" processing cores on FPGA. Software demo, *5th Conference on Real Numbers and Computers (RNC'5)*, Lyon, France, Sept. 2003.

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

[9] 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.

[10] G. Govindu, L. Zhuo, S. Choi, and V. Prasanna. Analysis of high-performance floating-point arithmetic on fpgas. 2004.

[11] E. G. W. III, M. J. Schulte, and M. G. Arnold. Truncated squarers with constant and variable correction. volume 5559, pages 40–50. SPIE, 2004.

[12] J. Liang, R. Tessier, and O. Mencer. Floating Point Unit Generation and Evaluation for FPGAs. *Field-Programmable Custom Computing Machines, Annual IEEE Symposium on*, 0:185, 2003.

[13] 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. ACM G.1.0; G.1.2; G.4; B.2.0; B.2.4; F.2.1., ISBN 978-0-8176-4704-9.

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

[15] R. Scrofano, G. Govindu, and V. K. Prasanna. A Library of Parameterizable Floating-Point Cores for FPGAs and Their Application to Scientific Computing. In T. P. Plaks, editor, *ERSA*, pages 137–148. CSREA Press, 2005.

[16] S. Srinath and K. Compton. Automatic generation of high-performance multipliers for fpgas with asymmetric multiplier blocks. In *FPGA '10: Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*, pages 51–58, New York, NY, USA, 2010. ACM.

[17] K. Underwood. FPGAs vs. CPUs: trends in peak floating-point performance. In *FPGA '04: Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pages 171–180, New York, NY, USA, 2004. ACM.

[18] F. de Dinechin, M. Joldes, and B. Pasca. Automatic generation of polynomial-based hardware architectures for function evaluation. LIP Research Report 2010-14, 2010.

[19] K. E. Wires, M. J. Schulte, and D. McCarley. FPGA Resource Reduction Through Truncated Multiplication. In *FPL '01: Proceedings of the 11th International Conference on Field-Programmable Logic and Applications*, pages 574–583, London, UK, 2001. Springer-Verlag.