Resource Optimal Truncated Multipliers for FPGAs

Andreas Böttcher, Martin Kumm, Florent de Dinechin

Resource Optimal Truncated Multipliers for FPGAs

Andreas Böttcher*, Martin Kumm*, Florent de Dinechin†
*Fulda, University of Applied Science, Fulda, Germany
†Univ Lyon, INSA Lyon, Inria, CITI, Lyon, France
andreas.boettcher@cs.hs-fulda.de, martin.kumm@cs.hs-fulda.de, florent.de-dinechin@insa-lyon.fr

***Abstract*—This proposal presents the resource optimal design of truncated multipliers targeting field programmable gate arrays (FPGAs). In contrast to application specific integrated circuits (ASICs), the design for FPGAs has some distinct design challenges due to many possibilities of computing the partial products using logic-based or DSP-based sub-multipliers. To tackle this, we extend a previously proposed tiling methodology which translates the multiplier design into a geometrical problem: the target multiplier is represented by a board that has to be covered by tiles representing the sub-multipliers. The tiling with the least resources can be found with integer linear programming (ILP). Our extension considers the error of possibly unoccupied positions of the board and determines the tiling with the least resources that respects the maximal allowed error bound. This error bound is chosen such that a faithfully rounded truncated multiplier is obtained. Compared to previous designs that use a fixed number of guard bits or optimize at the level of the dot diagrams, this allows a much better use of sub-multipliers resulting in significant area savings without sacrificing the timing.**

***Index Terms*—truncated multiplier, faithful rounding, field programmable gate arrays, multiplier tiling, integer linear programming, computer arithmetic**

I. Introduction

Multiplication is one of the most essential operations in computer arithmetic, so its resource efficient implementation is crucial for the economic design of complex circuits. The result of an exact $w_X \times w_Y$ multiplication is a number of $w_X + w_Y$ bits, but using only exact multipliers in an application would entail that the size of the internal data-paths grows along with the computation. Hence, the results of the multiplications are often truncated or rounded to an output word size of $w_P < w_X + w_Y$. An efficient way to do so is to perform faithful rounding, where the w_P most significant bits of the product P are computed, such that its absolute error bound is strictly smaller than its unit in the last place (ulp). While the design of truncated multipliers for application specific integrated circuits (ASICs) is a well studied field [1]–[6], the results can not be directly transferred to field programmable gate arrays (FPGAs). On FPGAs, the design has to be mapped into the logic resources or digital signal processing (DSP) blocks present on modern FPGAs.

The DSP units can be directly used to perform multiplications up to their nominal size. However, they are a limited resource and many applications necessitate calculations with precisions that either leave them underutilized, or require them to be completed with logic.

Much previous work covers the efficient realization of exact multipliers on FPGAs [7]–[21]. They can be divided into logic-based multipliers [9], [11]–[13], [17], [18], DSP-based multipliers [14], [19], logic/DSP hybrid methods [7], [8], [15], [16], [21] and the efficient summation of partial products in compressor trees [10], [22], [23].

In [9], the look-up-tables (LUTs) in conjunction with the fast carry chain of modern FPGAs were used to implement an efficient Baugh-Wooley multiplier. An efficient mapping for Xilinx FPGAs of a Booth multiplier was proposed in [11]–[13]. In [18], [19], an efficient LUT mapping for small multipliers uses the fractal synthesis approach. To reduce the under-utilization of DSPs, schemes to fit two smaller multiplications in one DSP were studied in [14], [19], [20]. The Karatsuba algorithm was used in [7], [16], [21] to trade DSPs for additional logic-based adders or subtractors and a longer critical path.

One generic attempt to handle logic/DSP hybrid designs in a generic way was introduced by the multiplier tiling methodology, first proposed in [7] and improved in [8], [15], [21]. It represents a large multiplier of arbitrary size as a geometric *board*, and each sub-multiplier producing a partial product as a *tile* that covers the board. The underlying tiling problem requires each position of the board to be covered by exactly one sub-multiplier tile. The sub-multiplier tiles are selected from a set of logic-based or DSP-based multipliers available on the target architecture.

Several methods were proposed to solve the multiplier tiling problem. In [8] and [10], heuristics were presented to cover the multiplier area with DSPs until a user-defined threshold is reached and to fill the remaining area with logic-based multipliers. While [8] use generic logic multipliers, more efficient 3×3-multipliers that map efficiently to the 6-input LUTs in recent FPGAs are utilized in [10]. A first optimal solution regarding the area-cost of the sub-multipliers was proposed in [15] by using integer linear programming (ILP). In [21], the greedy and beam search meta-heuristics were used to find solutions for the tiling problem for large multipliers (>64 bits) with an extensive set of logic-based and DSP-based sub-multipliers including those obtained from Karatsuba's algorithm.

Regarding truncated multipliers, most works so far target ASICs [1]–[6]. Just truncating the multiplier array always returns a value smaller than the exact result, therefore a first idea

[1], [2] is to add to the truncated array a constant correction that centers the error. Apart from the worst case absolute error, the mean square error can be further reduced by considering the replacement of this constant with a variable term that is a simple logic function of some of the discarded partial products [1], [3]–[5]. Most of these previous works present truncation as a trade-off between multiplier cost and accuracy, and the rich literature on approximate multiplier design [24] follows this path. Conversely, [6] starts with a well-accepted accuracy constraint (faithful rounding) which enables an ILP-based optimal design method for several truncated array multipliers like radix-2 and radix-4 Booth. The present article follows this approach, and extends it for FPGAs.

Fewer studies exist for truncated multipliers on FPGAs. An FPGA-specific truncated compression that uses a fixed number of guard bits was followed in [10]. The work in [8] considers a truncated multiplier as a non-rectangular tiling board with reduced area that has to be covered by sub-multipliers. The shape of this board is defined by a number of guard bits that allows faithful rounding. However a valid tiling forces the use of many small and inefficient sub-multipliers in the border region. A suggestion in [8] was to allow tiles to be placed crossing the border as defined by the guard bits and to use the additional precision to be able to compensate for allowing tiles to be omitted elsewhere, but no optimal solution is provided. The ILP formulation presented here extends [15] to dynamically consider the error of a tiling, thus ensuring that the tiling with the least cost satisfying the maximal allowed error bound is found. With that, more efficient larger sub-multipliers that may overlap the the border area can be placed, where the overlapping provides an additional error margin to omit multipliers elsewhere. The proposed ILP formulation provides significant savings over previous implementations and scales well to practically relevant sizes. Due to the elementary nature of the multiplication operation, many applications (e.g., from digital signal processing or machine learning) that do not require the the full precision of the results can benefit form the reduced realization costs.

II. Background

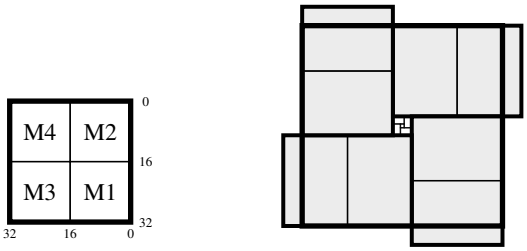
This section provides the necessary background about multiplier tiling and faithful rounding.

A. Multiplier Tiling

A large $X \times Y$-multiplier with input vectors X and Y of size $2W$ can be divided into two sub-words x_0, x_1 and y_0, y_1 of size W and the large $2W \times 2W$ bit multiplication can be substituted by four $W \times W$ multiplications $M_{1,..,4}$ as follows.

$$\begin{aligned} X \times Y &= \left(x_1 2^W + x_0\right)\left(y_1 2^W + y_0\right) \\ &= \underbrace{x_1 y_1}_{M_4} 2^{2W} + (\underbrace{x_1 y_0}_{M_3} + \underbrace{x_0 y_1}_{M_2}) 2^W + \underbrace{x_0 y_0}_{M_1} \end{aligned} \quad (1)$$

The result can be graphically represented as a square board of size $W \times W$, which is tiled by smaller rectangles representing the smaller multipliers [7]. The tiling of the example above is shown in Fig. 1a for a 32×32 multiplication using $W = 16$ bit multipliers.



(a) 32×32 mult. example (b) 53×53 mult. [15]

Fig. 1: Tilings for realizing different multipliers

This can be generalized, such that the large multiplier can be subdivided into smaller multipliers with arbitrary shapes and sizes [7]. In the FPGA context, the small multipliers are either realized using DSP blocks or as logic-based multipliers. The corresponding bit shifts can be directly read out from this graphical representation. A multiplier placed at position (x, y) has to be shifted by $x + y$ bits to the left in the final sum. For example, the multiplier $M4$ in Fig. 1a is located at coordinates $(16, 16)$, hence, its result has to be shifted by $16 + 16 = 32$ bits, which is also the result obtained from (1).

Fig. 1b shows the optimal tiling solution of the example of an unsigned 53×53 bit multiplier as required in a double precision multiplication using eight DSP blocks [15]. Here, the gray rectangles correspond to the 24×17-bit unsigned multiplications available in the Xilinx DSP48E(1) blocks while the white squares are realized in logic. As illustrated by this example, obtaining the tiling is a non-trivial task.

To formally model the tiling problem we introduce a couple of variables in the following. The shape and size of the desired multiplier is defined by a set of binary variables $M_{x,y}$, true for every position (x, y) within the area of the large multiplier to be covered, and otherwise false.

The set of available sub-multiplier tile shapes is $\mathcal{S} = \{m_0, m_1, ..m_{s-1}\}$. Each tile m_s is characterized by its realization costs in terms of utilized LUTs $\text{cost}^s_{\text{LUT}}$ and DSPs $\text{cost}^s_{\text{DSP}}$. Besides these costs, each tile m_s features a set of binary constants $m^s_{x,y}$ that describe its shape: $m^s_{x,y}$ is true for each position (x, y) covered by the tile s (relative to the tile origin $(0, 0)$), and false for every position outside of the tile. This approach allows arbitrarily shaped tiles. The set $\mathcal{S}$ used in this work is reviewed in Section II-B

The overall tiling problem can now be defined as follows. *Multiplier Tiling Problem: Given a shape $M_{x,y}$ of the large multiplier and a set $\mathcal{S}$ of sub-multipliers, each associated with cost$_s$, find a tiling with minimal cost such that each position (x, y) for which $M_{x,y} = 1$ is covered by exactly one instance of a sub-multiplier $m_s \in \mathcal{S}$.*

The cost function cost_s has to be compiled of a linear combination of $\text{cost}^s_{\text{LUT}}$ and $\text{cost}^s_{\text{DSP}}$. Alternatively, all multi-objective solutions can be obtained by minimizing $\text{cost}^s_{\text{LUT}}$ for a given number of DSP blocks.

TABLE I: Properties of LUT- and DSP-based multipliers [21]

Shape	Tile area	$\text{cost}^s_{\text{LUT}}$	$\text{cost}^s_{\text{DSP}}$
1×1	1	1.65	0
1×2	2	2.3	0
2×3	6	6.25	0
3×3	9	9.9	0
2×k	2k	1.65k + 2.3	0
24×17	408	26.65	1

B. Possible Sub-Multipliers

There are several options to select the tiles for the sub-multipliers. We list here the state-of-the-art for Xilinx FPGAs [21] that was also used to produce the results in this work. The possible sub-multiplier tiles are summarized in Table I with their shape, the area they cover on the board as well as their costs. The LUT costs consists of the number of LUTs that are required to perform the multiplication and the average number of LUTs that are required for the compressor tree ($\text{cost}^{\text{comp}}_{\text{LUT}}$). For our target FPGA, this was experimentally obtained to be $\text{cost}^{\text{comp}}_{\text{LUT}} = 0.65$ LUT/bit [15].

The logic-based tiles are either 3×3 multiplier mapped into 6-input LUTs [10] or smaller 2×3 [12]. To ensure that any shape can be tiled, we also include 1×2 and 1×1 tiles. A very efficient sub-multiplier that is based on LUTs and the carry chain is the $2 \times k$ multiplier. It consists basically of two rows of a Baugh-Wooley multiplier. We use the efficient mapping proposed in [9] here. The last row in Table I corresponds to a single DSP block.

C. Faithful rounding

Any multiplier that returns a result P on fewer than $w_X + w_Y$ bits must be inexact. Its error δ_{mult} with respect to the exact product $X \times Y$ is defined as

$$P = X \times Y + \delta_{\text{mult}} \quad . \tag{2}$$

Let l_P be the bit position of the least significant bit (LSB) of the returned product P (see Fig. 2). Faithful rounding corresponds to the constraint

$$|\delta_{\text{mult}}| < 1\,\text{ulp}(P) = 2^{l_P} \quad . \tag{3}$$

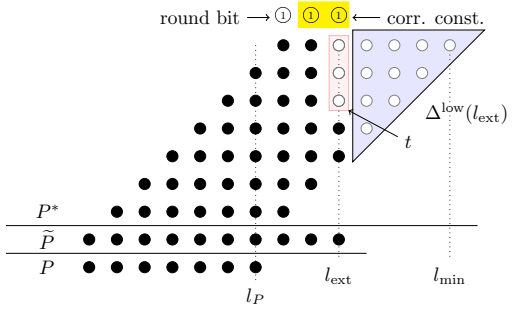


Fig. 2: Partial products of a faithfully rounded truncated multiplier for $w_X = w_Y = w_P = 7$, with $g = 3$ guard bits and $t = 3$ truncated bits in position l_{ext}

As Fig. 2 illustrates, a truncated multiplier first computes an approximate product $\widetilde{P}$ out of a truncated bit array, then rounds this $\widetilde{P}$ to obtain P. Each of these steps contributes to the overall error δ_{mult}:

$$\delta_{\text{mult}} = P - X \times Y \tag{4}$$

$$= \underbrace{P - \widetilde{P}}_{=\delta_{\text{round}}} + \underbrace{\widetilde{P} - X \times Y}_{\delta_{\widetilde{P}}} \tag{5}$$

Rounding $\widetilde{P}$ to the nearest value on the output format can be efficiently performed by truncating $\widetilde{P} + \frac{1}{2}\text{ulp}(P)$: all it takes is to add a constant 1 of weight 2^{l_P-1} in the compressor tree (round bit in Fig. 2). The corresponding rounding error verifies $|\delta_{\text{round}}| \leq 2^{l_P-1}$. To comply with the requirement (3), the approximate product error then has to fulfill:

$$|\delta_{\widetilde{P}}| < 2^{l_P-1} \quad . \tag{6}$$

This error is maximal if all of the omitted partial products are equal to 1 in the full bit array.

In the non-truncated multiplier, those partial products are calculated as

$$P^* = X \times Y = \sum_{j=0}^{w_Y-1} \sum_{i=0}^{w_X-1} 2^{i+j} x_i y_j \ . \tag{7}$$

The core idea when implementing a faithfully rounding truncated multiplier is to remove as many partial products from the right of the resulting compressor tree without violating the multiplication error constraint as defined by (6). Omitting a partial product in compressor tree column $i + j$ contributes 2^{i+j} to the error $\overline{\delta}_{\text{mult}} = \widetilde{P} - P^*$ according to (7). To meet the constraint (6) partial products can only be removed in columns smaller than $2^{i+j} < 2^{l_P-1}$. It is generally advantageous to start removing partial product bits in the least significant columns of the compressor tree: for the same error contribution, removing two bits with weight 2^l is better than removing one with weight 2^{l+1}, as the cost for calculating each bit is about the same. The challenge is now to estimate the largest column index $l_{\text{ext}} = l_P - g$ and the largest number t of truncated bits in column l_{ext} to respect the error bound of (6). For this purpose let us define $\Delta(l_{\text{ext}}, t)$ the resulting maximal error (when all bits in the exact result P^* would be set) as follows:

$$\Delta(l_{\text{ext}}, t) = t\,2^{l_{\text{ext}}} + \underbrace{2^{l_{\text{min}}}\left(1 + 2 \cdot 2^1 + 3 \cdot 2^2 ... + n \cdot 2^{n-1}\right)}_{=\Delta^{\text{low}}(l_{\text{ext}})} \tag{8}$$

with $n = l_{\text{ext}} - l_{\text{min}}$ (see Fig. 2). This weighted sum of the truncated bits can be simplified by applying the evaluated sum formula of the recurring products

$$\sum_{i=0}^{n-1} (i+1)\,2^{i+j} = (n-1) \cdot 2^n + 1 \ , \tag{9}$$

so $\Delta^{\text{low}}(l_{\text{ext}})$ becomes

$$\Delta^{\text{low}}(l_{\text{ext}}) = 2^{l_{\text{min}}}\left((n-1) \cdot 2^n + 1\right) \ . \tag{10}$$

Because truncation removes partial product bits from the initial compressor tree, the resulting error $\delta_{\widetilde{P}} = \widetilde{P} - P^*$ is negative: $\delta_{\widetilde{P}} \in [-\Delta, 0]$. As suggested in [2] the error $\overline{\delta}_{\widetilde{P}}$ can be halved by adding a constant $C = \Delta/2$ to the compressor tree, to statistically center the error around zero $\delta_{\widetilde{P}} \in [-\Delta/2, \Delta/2]$. But because the absolute error is still limited by $\overline{\delta}_{\widetilde{P}} < 2^{l_P-1}$, the relative tiling error $\delta_{\widetilde{P}}$ can be twice as large, which can be utilized to omit further partial products. In practice, C should have its individual bits c_i between positions $l_P - 2$ (since values above would violate the error constraint), down to l_{ext} (because the extension of the compressor tree with columns with lower weight is mostly undesirable). To maximize truncation, we set C as the maximal value that fits on these bits:

$$C = 2^{l_P-1} - 2^{l_{\text{ext}}} \tag{11}$$

With this constant C added to the array, the error now resides in the interval

$$\delta_{\widetilde{P}} \in [C - \Delta, C]. \tag{12}$$

The upper bound by definition verifies $C < 2^{l_P-1}$. The absolute value of the (negative) lower bound is deduced from (8):

$$\Delta(l_{\text{ext}}, t) - C = t \cdot 2^{l_{\text{ext}}} + \Delta^{\text{low}}(l_{\text{ext}}) - C \tag{13}$$

$$= t \cdot 2^{l_{\text{ext}}} + \Delta^{\text{low}}(l_{\text{ext}}) - \left(2^{l_P-1} - 2^{l_{\text{ext}}}\right) \tag{14}$$

As this negative component must also respect (6), (14) becomes (after some rearranging):

$$(t+1) \cdot 2^{l_{\text{ext}}} + \Delta^{\text{low}}(l_{\text{ext}}) < 2^{l_P} \tag{15}$$

The goal is now to calculate the maximal value of l_{ext}, and the maximal number t of bits that may be truncated in column l_{ext}. Those parameters can be calculated iteratively in linear time as shown in Algorithm 1, as as function of l_P or, equivalently, of the number of skipped bits with regard to the exact result $w_{skip} = w_{full} - w_P$. The algorithm consists of two loops: in the first loop, the number of truncated columns is increased as long as (15) is satisfied. When this is the case, position l_{ext} is found, and therefore also the corresponding constant C given by (11). The second loop increases the number of truncated bits t again as long as (15) is satisfied. It turns out that this simple greedy optimization process provides the sames result as the ILP of [6]. Also note that it can be used for other multiplier variants such as Booth multipliers, squarers, etc: all it takes is an implementation of the computation of $\Delta^{\text{low}}(l_{\text{ext}})$ out of the corresponding bit array.

D. Truncated Multipliers using Existing Tiling Methods

We will now discuss how the observations in the previous section can be transferred into a tiling problem that can be used by existing tiling methods. This is done by using a modified tiling board, where the function $M_{x,y}$ defining the multiplier shape is slightly altered: All position (x, y) having a weight $2^{x+y} < 2^{l_{\text{ext}}}$ are removed, resulting in a tiling board where the upper right part is missing (see Fig. 3).

Algorithm 1: Algorithm to calculate truncation parameters

```
// Truncation of a multiplier faithful to 2^lP
computeTruncation(lP):
  // initialization
  lext ← lmin // start from the right
  t ← 0 // no bit is removed

  // First loop: determine lext
  while ( (t+1)·2^(lext+1) + Δ^low(lext+1) < 2^lP )
    lext ← lext + 1

  // Second loop: determine t
  while ( (t+2)·2^lext + Δ^low(lext) < 2^lP )
    t ← t + 1

  return (lext, t)
```

In addition, we may truncate t bits in column l_{ext}. These are located in the diagonal with weight $2^{l_{\text{ext}}}$. Hence, we can remove an arbitrary selection of t positions on the board with weight $2^{l_{\text{ext}}}$. This is illustrated by the red crosses in Fig. 3. In our heuristic reference implementation we start linearly from the top to create a cohesive area.

III. Existing Multiplier Tiling Formulation

We now describe the existing ILP formulation for the multiplier tiling of [15]. We will later extend the ILP formulation to directly design truncated multipliers.

The first constraint of the ILP formulation describes the possible placement of the multipliers on the area to be tiled. For this, a binary decision variable $d^s_{x,y}$ is introduced for each shape and position, that is true if a sub-multiplier tile of shape s is placed at position (x, y). Now, a valid tiling has to fulfill the following constraints:

$$\text{C1:} \quad \sum_{s=0}^{S-1} \sum_{x'=0}^{x} \sum_{y'=0}^{y} m^s_{x-x',y-y'} \, d^s_{x',y'} = 1 \quad \left.\begin{array}{l} \text{for } 0 \le x < w_X, \\ 0 \le y < w_Y \\ \text{with } M_{x,y} = 1 \end{array}\right\}$$

The constraints make sure that every possible position (x, y) within $0 \le x < w_X$ and $0 \le y < w_Y$ is covered by exactly one tile.

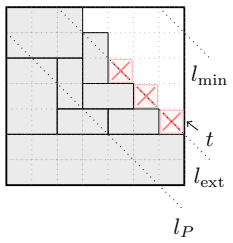
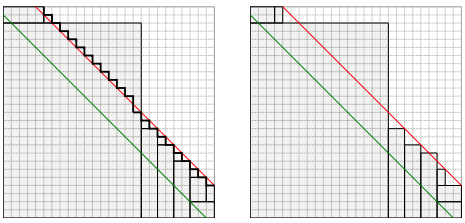


Fig. 3: Tiling area of a faithfully rounded truncated multiplier for $w_X = w_Y = w_P = 7$, with $g = 3$ and $t = 3$



(a) fixed border (b) dynamic

Fig. 4: Tilings for a 26×26-multiplier with $w_P = 26$, showing l_{ext} (red), l_P (green) and fixed border (black)

The objective is now to minimize the cost function, which is simply the sum of all the decision variables $d^s_{x,y}$ weighted with their cost contribution cost^s

$$\text{minimize} \sum_{s=0}^{S-1}\sum_{x=0}^{X-1}\sum_{y=0}^{Y-1} \text{cost}^s_{\text{LUT}} d^s_{x,y} \ . \tag{16}$$

To obtain the minimal LUT cost for a given number of DSPs, we set $\text{cost}^s = \text{cost}^s_{\text{LUT}}$ and add a second constraint to limit the DSPs

$$\text{C2:} \qquad \sum_{s=0}^{S-1}\sum_{x=0}^{X-1}\sum_{y=0}^{Y-1} \text{cost}^s_{\text{DSP}} d^s_{x,y} \leq \#\text{DSP} \ .$$

IV. Proposed truncated multiplier ILP formulation

Applying the previous tiling solution to the truncated board as described above delivers valid truncated multipliers but may not lead to optimal ones. Fig. 4a shows an example tiling of a 26x26-multiplier truncated to $w_P = 26$ output bits. First, there may be tiles that largely overlap like the large DSP tile. Hence, the error is less than expected which can be exploited to allow a larger error at different positions. Second, the diagonal border requires many of the 1×1 tiles which are the least effective tiles. Here, the exact location of the t bits to be truncated in the rightmost column is missing. This location is irrelevant for ASIC multipliers, where all bits are created equal out of AND gates. In a tiling-based multiplier for FPGAs, however, this location matters: as Fig. 4 shows, a rectangular tile that extends to the right of the truncation column strongly constrains the location of the truncated bits in the rightmost column. Besides, it greatly reduces $\Delta^{\text{low}}(l_{\text{ext}})$, to the point that sometimes it will be possible to truncate bits to the left of column l_{ext}.

Hence, it appears beneficial, rather than defining a fixed border, to consider the resulting error directly in the ILP formulation. Thus the cost optimal tiling for a given error bound can be obtained.

To allow positions to remain uncovered, the coverage constraint C1 of the previous ILP formulation has to be modified. Instead of setting the right hand side of constraint C1 to 1, it is set to a new binary variable $b_{x,y}$:

$$\text{C1'}: \sum_{s=0}^{S-1}\sum_{x'=0}^{x}\sum_{y'=0}^{y} m^s_{x-x',y-y'} d^s_{x',y'} = b_{x,y} \left.\begin{array}{l} \text{for } 0 \leq x < w_X, \\ 0 \leq y < w_Y \\ \text{with } M_{x,y} = 1 \end{array}\right\}$$

While $b_{x,y} = 1$ signifies no error, leaving a position uncovered $b_{x,y} = 0$ introduces a negative error, since the resulting approximate product $\widetilde{P}$ is less or equal then the exact product P^*, due to the missing contribution of the partial product at (x, y). Hence, the resulting error that results from omitting position (x, y) can be quantified as

$$\overline{\delta}_{x,y} = (1 - b_{x,y}) \cdot 2^{x+y} \ . \tag{17}$$

The resulting total tiling error of every skipped partial product is:

$$\overline{\delta}_{\text{tiling}} = \sum_{x=0}^{X-1}\sum_{y=0}^{Y-1} \overline{\delta}_{x,y} \tag{18}$$

The actual error δ_{tiling} is bound by $0 \leq \delta_{\text{tiling}} \leq \overline{\delta}_{\text{tiling}}$ and has a negative contribution to the total error. Hence, we can reduce the absolute error by adding a constant C to the compressor tree to recenter the negative error around zero as proposed in previous work [1], [2]. The total tiling error constraint including the constant error becomes

$$|C - \overline{\delta}_{\text{tiling}}| = \overline{\delta}_{\widetilde{P}} < 2^{l_P - 1} \ . \tag{19}$$

To utilize this constraint in the ILP formulation it has to be linearized by considering the following two cases: 1) When the tiling error is smaller then the constant, i.e., $\delta_{\text{tiling}} \leq C$, this results in a positive total error that becomes maximal when $\delta_{\text{tiling}} = 0$. So, to meet the faithfulness requirements the constant C has to be constraint to:

$$\text{C2a:} \qquad C < \delta_{\widetilde{P}}$$

2) When $C < \delta_{\text{tiling}}$, the error (19) is negative and becomes maximal when $\delta_{\text{tiling}} = \overline{\delta}_{\widetilde{P}}$. In this case the error has to be bound to:

$$C - \overline{\delta}_{\text{tiling}} > -\overline{\delta}_{\widetilde{P}} \tag{20}$$

By inserting (17) and (18), (20) can be reformulated as

$$\text{C2b:} \qquad \sum_{x=0}^{X-1}\sum_{y=0}^{Y-1} (1 - b_{x,y}) \cdot 2^{x+y} - C < \overline{\delta}_{\widetilde{P}} \tag{21}$$

The constant C improves the error but adds additional bits to the compressor tree. To account for this, the ILP formulation should also be capable of dynamically altering the constant C and considering its cost. To do so, we describe it as the weighted sum of its individual bits c_i using the constraint

$$\text{C3:} \qquad C = \sum_{i=l_{\text{ext}}}^{l_P-2} 2^i c_i \ . \tag{22}$$

Each bit is described as a binary variable c_i. The bits are considered in the model from below $l_P - 2$ the weight of the rounding constant (at $l_P - 1$) down to the lowest realized column with partial products l_{ext}. Constant bits with larger weight violate (19) while constant bits with lower weight are less beneficial since the gain in error margin falls with its weight but the cost for compressing the constant bit remains equal.

The objective has to be extended to accommodate for the individual bits c_i

$$\text{minimize} \sum_{s=0}^{S-1} \sum_{x=0}^{X-1} \sum_{y=0}^{Y-1} \text{cost}_{\text{LUT}}^{s} d_{x,y}^{s} + \sum_{i=l_{\text{ext}}}^{l_P-2} \text{cost}_{\text{LUT}}^{\text{comp}} c_i \ . \quad (23)$$

Remember that the term $\text{cost}_{\text{LUT}}^{\text{comp}}$ denotes the average LUT cost for one bit that has to be compressed in the compressor tree. The constraint of the previous ILP formulation to limit the DSPs (C2) remains unchanged.

To accelerate the ILP solving process, positions (x, y) with a weight larger then l_P are set to $b_{x,y} = 1$, since those positions would already violate the error constraint.

Fig. 4b shows the tiling result when using the extended ILP formulation on the same problem as used in Fig. 4a. Due to the contribution of the DSP block, a smaller overall area has to be covered. Besides, the efficient $2 \times k$ can be better utilized, completely avoiding the use of costly 1×1 tiles.

V. Results

A. Experimental Setup

The ILP formulations of the tiling problems are implemented by extending the existing `IntMultiplier` operator of the open source VHDL code generator framework FloPoCo [25]. As ILP solver, we used Gurobi 8.1.1. Xilinx Vivado 2019.1 was used for the synthesis experiments targeting a Kintex-7 FPGA (xc7k70tfbv484-3) and incorporated place&route. The timing data was obtained by evaluating the critical path between wrapper registers placed at the inputs and outputs of the circuit, to achieve a realistic application scenario where the multiplier is integrated in a larger design.

B. Conducted Experiments

We consider the existing ILP formulation presented in Section III applied to the modified tiling board for truncated multipliers of Section II-D as the state-of-the-art for truncated faithfully rounded multipliers on FPGAs and refer it to "fixed border". The proposed ILP extension of Section IV is referred to as "dynamic border". A set of experiments was devised to compare both methods for practical use cases. In the first test series, multipliers for the common case $w_X = w_Y = w_P$ were generated for word sizes between 2 and 32. To also evaluate the performance of the method subject to different truncations, multipliers with a fixed size of 32×32 and varying output sizes w_P between 1 and 64 were generated in a second test series. Since a single DSP already covers most of the area of the examined multipliers, both test series were repeated with 0 and 1 allowed DSP, to separately assess the performance with small LUT-based multiplier tiles and DSPs.

C. Area Results

Fig. 5 plots, for the first test series with 0 DSPs, the objective functions from (16) and (23) along with the corresponding synthesis results. As expected, the objective shows an improvement in every case. This improvement increases with the size of the multiplier, following the increase of the border area where the new method can yield a more favorable placement of larger, more efficient LUT multipliers. The synthesis results in utilized LUTs generally follow this trend, although the deviation between the modeled and the actual compression cost introduces noise in the synthesis results. The relative difference is fluctuating the most in the smaller cases, since a single LUT corresponds to a large relative change.

The next experiment with a 32×32 multiplier and $w_P = 1..64$ shown in Fig. 6(a) exhibits an s-shaped curve in regards to the LUT cost, because the change in covered area and hence partial products becomes maximal at the middle output width $w_P = 32$ due to the quadratic tiling area. Accordingly, as shown in Fig. 6(b) the absolute improvement also becomes maximal at $w_P = 32$. As expected, the absolute improvement of the synthesis results show the same trend as the improvement of the objective but are a bit more noisy.

The relative improvement decreases for larger output sizes, due to the large increase in total circuit complexity, especially beyond the largest improvements around $w_P = 32$.

So far, the improvements were rather modest but the main advantage appears when using additional DSP blocks. Hence, the first experiment with equal input and output sizes between 2..32 was repeated between 16..32 with 1 DSP as shown in Fig. 7, since below $w_X = w_Y = w_P = 17$ it would only consist of the DSP. The results show an improvement in every case, highlighting the benefits of the presented method. In the graphical tiling representation, the DSP block reaches in many cases far over the border defined by the fixed border method. The LSBs of the DSP block would traditionally remain unused. By incorporating those LSBs in the total error budget, the proposed method saves a considerable amount of LUT-based multipliers elsewhere.

The second experiment for a 32×32 multiplier and $w_P = 1..64$ was also repeated with 1 DSP, for $w_P = 16..64$ (Fig. 8). As in the test series with 0 DSP the improvements reach their maximum around $w_X = w_Y = w_P$ and the synthesis results follow the trend of the objective.

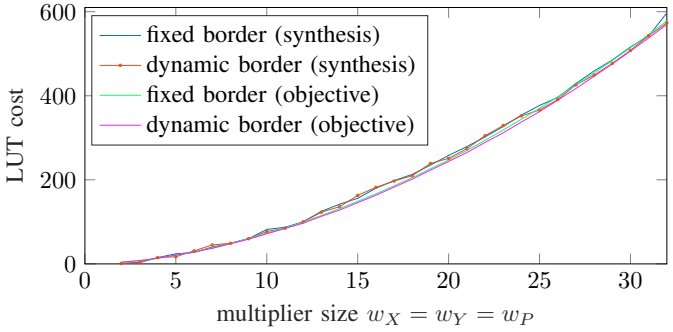
The solving time in all the experiments did never exceed 5 minutes.

D. Delay Results

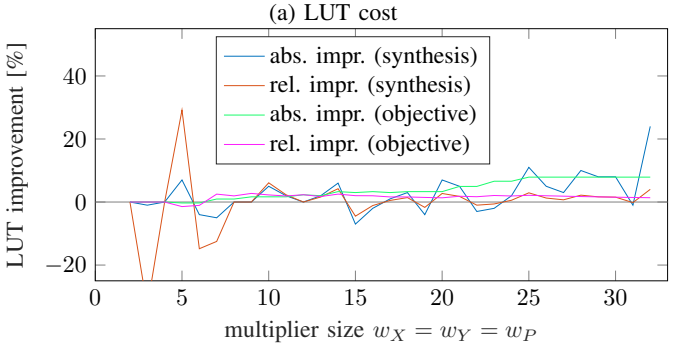
The timing results are shown in figures 9 and 10 and, as expected, do not show significant differences in the resulting critical path between the two methods. The limiting factor for the critical path is the net delay of the signal routing between the low-level resources of the FPGA. While the timing characteristics can be easily retrieved from the data sheet, the eventual placement of the components by Vivado introduces a degree of uncertainty, when performing the automatic pipelining in FloPoCo. This also explains the slight rise of the resulting critical path for larger multipliers, since the increase in complexity also raises the likelihood that expected logic delay is exceeded.

VI. Conclusion

This proposal demonstrates how the state-of-the-art in multiplier tiling methodology can be applied to build faithfully

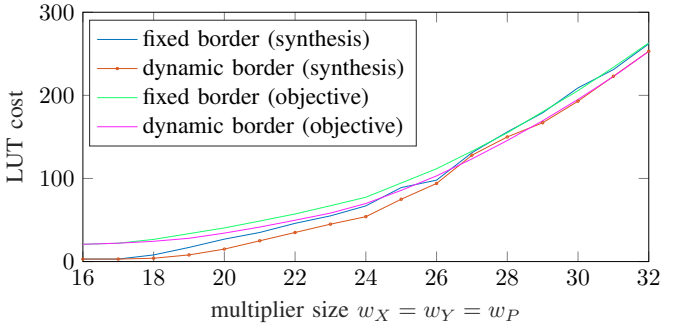


(a) LUT cost

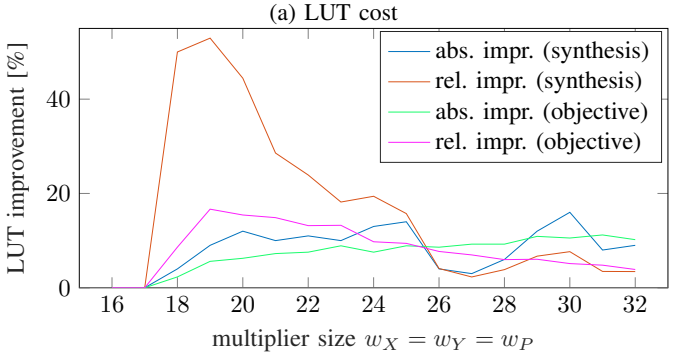


(b) improvement

Fig. 5: Results for multiplier sizes $w_X = w_Y = w_P = 2..32$ using 0 DSP (logic only)

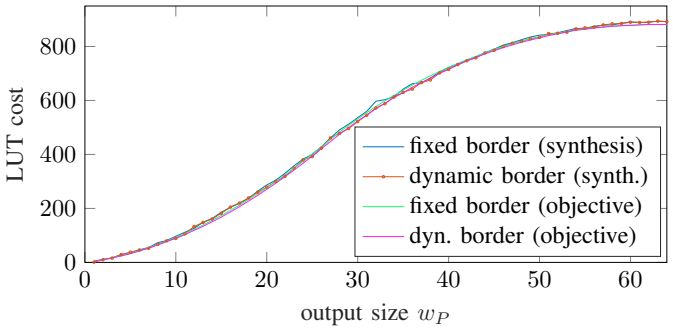


(a) LUT cost

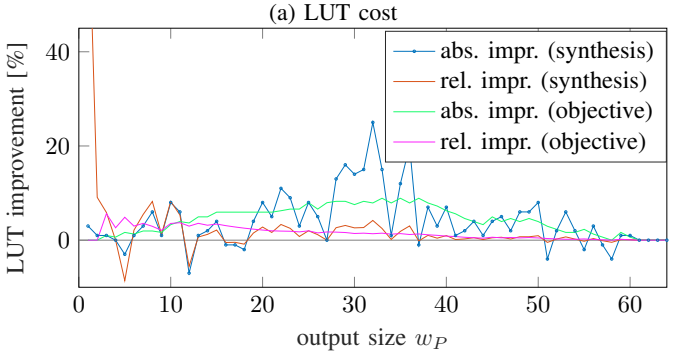


(b) improvement

Fig. 7: Results for multiplier sizes $w_X = w_Y = w_P = 2..32$ using 1 DSP

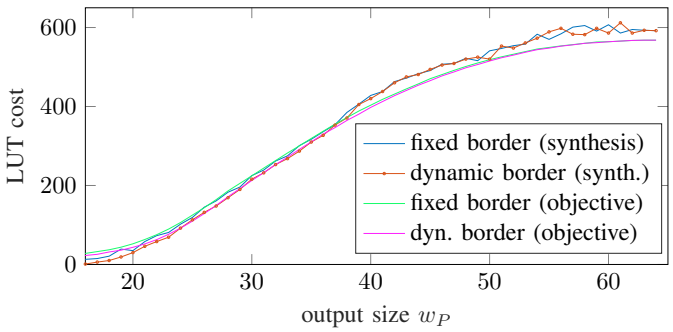


(a) LUT cost

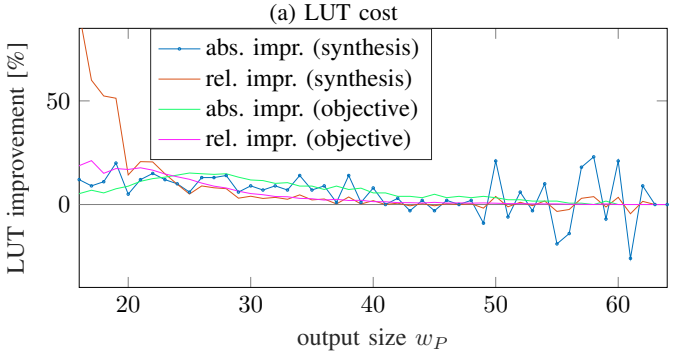


(b) improvement

Fig. 6: Results for a 32x32 multiplier with $w_P = 2..64$ using 0 DSP (logic only)



(a) LUT cost



(b) improvement

Fig. 8: Results for a 32x32 multiplier with $w_P = 2..64$ using 1 DSP

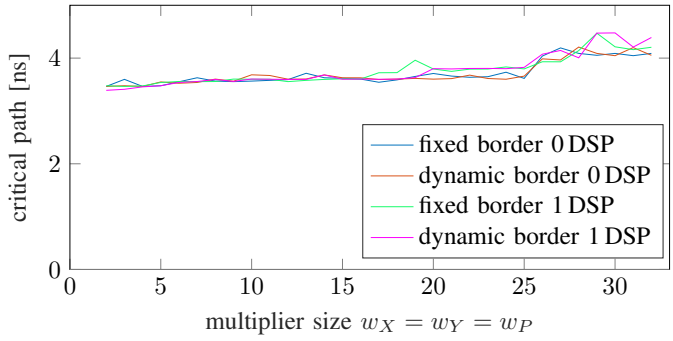


Fig. 9: Critical path delay for multiplier size $w_X = w_Y = w_P = 2..32$

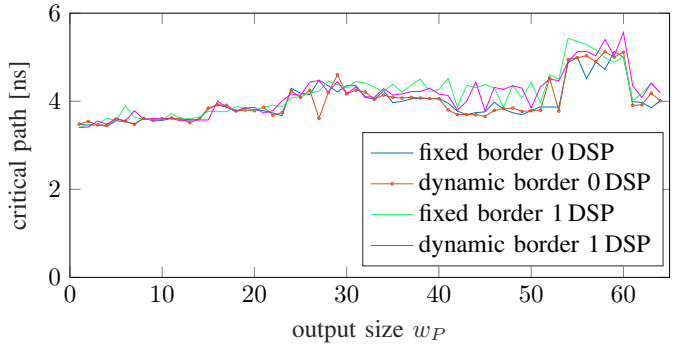


Fig. 10: Critical path delay for 32×32 multipliers $w_P = 16..64$ with 1 DSP

rounded truncated multipliers for FPGAs. It was shown that in many cases this baseline can be improved by extending the existing ILP formulation to constrain the tiling error instead of the tiled area and to consider the resulting error dynamically. This avoids the placement of relatively small multiplier tiles at the border of the tiled area in favor of larger, more efficient variants. When larger tile like DSPs overlap the former border, this can be used to compensate for leaving positions uncovered elsewhere. The results show that the larger the border area is, the larger are the savings, since there are more positions where potentially a more efficient tiling can be found.

REFERENCES

[1] Y. Lim, "Single-precision multiplier with reduced circuit complexity for signal processing applications," *IEEE Computer Architecture Letters*, vol. 41, no. 10, pp. 1333–1336, 1992.

[2] M. J. Schulte and E. E. Swartzlander, "Truncated multiplication with correction constant [for DSP]," in *Proceedings of IEEE Workshop on VLSI Signal Processing*, 1993, pp. 388–396.

[3] E. J. King and E. E. Swartzlander, "Data-dependent truncation scheme for parallel multipliers," in *Asilomar Conference on Signals, Systems and Computers*, vol. 2. IEEE, 1997, pp. 1178–1182.

[4] R. Michard, A. Tisserand, and N. Veyrat-Charvillon, "Carry prediction and selection for truncated multiplication," in *Workshop on Signal Processing Systems Design and Implementation*. IEEE, 2006, pp. 339–344.

[5] N. Petra, D. De Caro, V. Garofalo, E. Napoli, and A. G. Strollo, "Truncated binary multipliers with variable correction and minimum mean square error," *Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 6, pp. 1312–1325, 2009.

[6] T. A. Drane, T. M. Rose, and G. A. Constantinides, "On the systematic creation of faithfully rounded truncated multipliers and arrays," *IEEE Transactions on Computers*, vol. 63, no. 10, pp. 2513–2525, 2014.

[7] F. de Dinechin and B. Pasca, "Large multipliers with fewer DSP blocks," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2009, pp. 250–255.

[8] S. Banescu, F. de Dinechin, B. Pasca, and R. Tudoran, "Multipliers for floating-point double precision and beyond on FPGAs," *SIGARCH Comput. Archit. News*, vol. 38, no. 4, p. 73–79, Jan. 2011. [Online]. Available: https://doi.org/10.1145/1926367.1926380

[9] H. Parandeh-Afshar and P. Ienne, "Measuring and reducing the performance gap between embedded and soft multipliers on FPGAs," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2011, pp. 225–231.

[10] N. Brunie, F. de Dinechin, M. Istoan, G. Sergent, K. Illyes, and B. Popa, "Arithmetic core generation using bit heaps," in *International Conference on Field programmable Logic and Applications (FPL)*, 2013, pp. 1–8.

[11] E. G. Walters, "Partial-product generation and addition for multiplication in FPGAs with 6-input LUTs," in *Asilomar Conference on Signals, Systems and Computers*, 2014, pp. 1247–1251.

[12] M. Kumm, S. Abbas, and P. Zipf, "An efficient softcore multiplier architecture for Xilinx FPGAs," in *Symposium on Computer Arithmetic (ARITH)*, 2015, pp. 18–25.

[13] E. G. Walters, "Array multipliers for high throughput in Xilinx FPGAs with 6-input LUTs," *Computers*, vol. 5, no. 4, 2016. [Online]. Available: https://www.mdpi.com/2073-431X/5/4/20

[14] Y. Fu, E. Wu, A. Sirasao, S. Attia, K. Khan, and R. Wittig, "Deep learning with INT8 optimization on Xilinx devices (white paper)," Xilinx, Inc., Tech. Rep., 2017.

[15] M. Kumm, J. Kappauf, M. Istoan, and P. Zipf, "Resource optimal design of large multipliers for FPGAs," in *Symposium on Computer Arithmetic (ARITH)*, 2017, pp. 131–138.

[16] M. Kumm, O. Gustafsson, F. de Dinechin, J. Kappauf, and P. Zipf, "Karatsuba with rectangular multipliers for FPGAs," in *Symposium on Computer Arithmetic (ARITH)*, 2018, pp. 13–20.

[17] M. Langhammer and G. Baeckler, "High density and performance multiplication for FPGA," in *IEEE Symposium on Computer Arithmetic (ARITH)*, 2018.

[18] M. Langhammer, G. Baeckler, and S. Gribok, "Fractal synthesis: Invited tutorial," in *International Symposium on Field-Programmable Gate Arrays (FPL)*, ser. FPGA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 202–211. [Online]. Available: https://doi.org/10.1145/3289602.3293927

[19] M. Langhammer, B. Pasca, G. Baeckler, and S. Gribok, "Extracting INT8 multipliers from INT18 multipliers," in *International Symposium on Field-Programmable Gate Arrays (FPL)*, 2019, pp. 114–120.

[20] M. Langhammer, S. Gribok, and G. Baeckler, "High density 8-bit multiplier systolic arrays for FPGA," in *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2020, pp. 84–92.

[21] A. Böttcher, K. Kullmann, and M. Kumm, "Heuristics for the design of large multipliers for FPGAs," in *Symposium on Computer Arithmetic (ARITH)*, 2020, pp. 17–24.

[22] H. Parandeh-Afshar, A. Neogy, P. Brisk, and P. Ienne, "Compressor tree synthesis on commercial high-performance FPGAs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 4, no. 4, Dec. 2011. [Online]. Available: https://doi.org/10.1145/2068716.2068725

[23] M. Kumm and J. Kappauf, "Advanced compressor tree synthesis for FPGAs," *IEEE Transactions on Computers*, vol. 67, no. 8, pp. 1078–1091, 2018.

[24] H. Jiang, C. Liu, L. Liu, F. Lombardi, and J. Han, "A review, classification, and comparative evaluation of approximate arithmetic circuits," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 4, pp. 1–34, 2017.

[25] F. de Dinechin and B. Pasca, "Designing custom arithmetic data paths with FloPoCo," *IEEE Design & Test of Computers*, vol. 28, no. 4, pp. 18–27, 2012.