Small Logic-based Multipliers with Incomplete Sub-Multipliers for FPGAs

Andreas Böttcher, Martin Kumm
Fulda, University of Applied Sciences, Faculty of Applied Computer Science, Fulda, Germany
andreas.boettcher@cs.hs-fulda.de, martin.kumm@cs.hs-fulda.de

***Abstract*—There is a recent trend in artificial intelligence (AI) inference towards lower precision data formats down to 8 bits and less. As multiplication is the most complex operation in typical inference tasks, there is a large demand for efficient small multipliers. The large DSP blocks have limitations implementing many small multipliers efficiently. Hence, this work proposes a solution for better logic-based multipliers that is especially beneficial for small multipliers. Our work is based on the multiplier tiling method in which a multiplier is designed out of several sub-multiplier tiles. The key observation we made is that these sub-multipliers do not necessarily have to perform a complete (rectangular) $N \times K$ multiplication and more efficient sub-multipliers are possible that are incomplete (non-rectangular). This proposal first seeks to identify efficient incomplete irregular sub-multipliers and then demonstrates improvements over state-of-the-art designs. It is shown that optimal solutions can be found using integer linear programming (ILP), which are evaluated in FPGA synthesis experiments.**

***Index Terms*—small multiplier, tiling, computer arithmetic, FPGA,**

I. Introduction

With multiplication being one of the most pervasive operations in computer arithmetic, its efficient implementation is of ongoing interest. When targeting FPGAs the design methods have to consider the embedded DSP blocks and look-up-tables (LUTs) as the primary resources of modern devices. DSP blocks are usually the most convenient to use, as they allow the implementation of a multiplication by themselves (for example up to one 27×27 or two 18×18 signed or unsigned on Intel Stratix10 or 25×18 signed or 24×17 unsigned on AMD 7 Series) but are a relatively limited resource. Multiplier design methods have to accommodate for that to avoid the under-utilization or premature depletion of DSP resources when generating complex circuits [1]–[4]. A multitude of methods have been proposed to remedy the underlying design challenge [1]–[14].

In [9], [12], [13], [15] the extraction of smaller multipliers from one DSP by additional LUT-based circuitry was proposed. Conversely, when larger multipliers than the size of a single DSP are needed, Karatsuba's algorithm can be used to reduce the number of DSPs [1], [14]. With Slice LUTs being one of the most prolific resources on FPGAs, several concepts that efficiently use LUT resources in conjunction with the fast-carry-chain like implementations of vendor-specific Booth-Arrays [3], [7], [8], Baugh-Wooley-Multipliers [5] and several optimizations with the fractal-synthesis approach [10], [11] were devised.

Another design process is multiplier tiling [1] which seeks to achieve the efficient use of DSPs by combining them with supplementary LUT-based sub-multipliers. But as aside from heuristics [2], [6], [14] also optimal [4], [16] methods for solving the underlying tiling problem are known, in the sense that for a given set of sub-multipliers (tiles) the combination with the least realization cost in terms of logic resources is found while constraining the use of DSPs to a maximal number. Hence, to generate smaller designs as in the present proposal, where DSPs would be considerably underutilized, by constraining the number of DSPs to zero, multiplier tiling can be used to generate logic-only solutions which constitute the optimal use of LUT resources. Multipliers generated with tiling usually consist of the sub-multipliers (tiles) as partial product generators, which are subsequently compressed on a compressor tree to form a single output vector. The compressor tree design for FPGAs has to consider certain peculiarities in contrast to ASICs, as it has to be implemented within the confines of the considered target architecture. As full- or half-adders under-utilize the FPGA slice hardware, more efficient structures like generalized parallel counters (GPCs) [17]–[25], 4:2 row compressors [22], [26], [27] and the ternary adder [28], [29] were devised, which utilize the fast-carry-chain of modern FPGAs. The introduction of the additional compressors significantly extended the design space to be explored to find efficient solutions, which necessitates advanced concepts for compressor tree design beyond Dadda [30]. Several heuristic [18], [20], [21] and optimal [19], [23], [32]–[35] methods have been presented for this task. Since the selection of the partial product generators (tiles), compressor tree and final adder design are interdependent problems, as a different choice in one of the steps can result in an alternative combination becoming the the optimal solution in another, a recent combined approach [35] that optimizes both the tiling and the compressor tree simultaneously, is used in the present proposal.

Except for the Karatsuba algorithm with rectangular DSPs [36] and DSP super tiles [1] the previous mapping schemes usually aimed on creating rectangular (sub-)multipliers.

In AI inference tasks, it is commonly accepted that 8 bit integers are sufficient to reach floating point accuracy with a proper scaling and quantization-aware training in many applications. Recent transformer networks are typically quantized to even 4 bits. This motivates this work to focus on small multipliers.

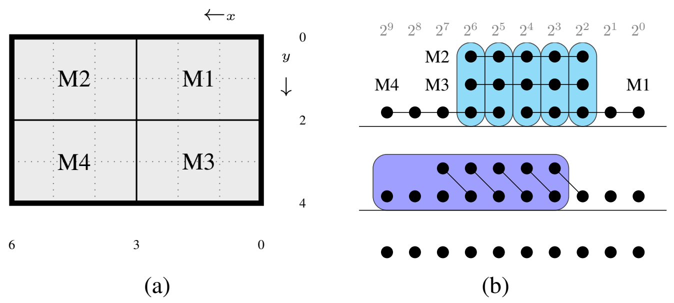


Fig. 1: Tiling (a) and compressor tree (b) of a 6×4-multiplier, composed of four 3×2-tiles.

The present proposal

- presents a novel class of incomplete logic-based (sub-) multipliers that map efficiently to most modern FPGAs,
- demonstrates the integration and use in conjunction with the multiplier tiling methodology to design complete multipliers,
- demonstrates that the proposed designs yield a reduction in utilized resources compared to other state-of-the-art logic-based multiplier design methods while maintaining similar properties in terms of critical path delay and latency.

II. Multiplier Tiling

Consider a large $2W_X \times 2W_Y$ sized multiplier. The $X \times Y$ multiplication operation can be decomposed into the weighted sum of four $W_X \times W_Y$ sub-multiplications $M_{1,..,4}$ according to

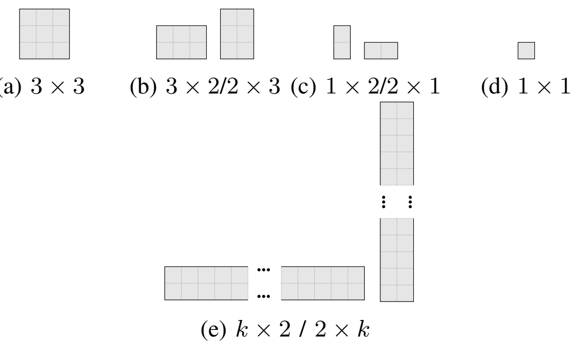
$$\begin{aligned} X \times Y &= \left(x_h 2^{W_X} + x_l\right)\left(y_h 2^{W_Y} + y_l\right) \\ &= \underbrace{x_h y_h}_{M_4} 2^{W_X+W_Y} + \underbrace{x_h y_l}_{M_3} 2^{W_X} + \underbrace{x_l y_h}_{M_2} 2^{W_Y} + \underbrace{x_l y_l}_{M_1} \, . \end{aligned} \tag{1}$$

Thereby the input vectors X and Y are split into a most and least significant sub-word x_h, x_l and y_h, y_l with a respective size of W_X and W_Y. Accordingly by splitting the input vectors X and Y into the individual bits, this process can be repeated down to the respective partial product bits, as they would be generated by the AND-gates of a Bough-Wooley multiplier. However, in practical tilings usually larger multipliers are used which map more efficiently to the hardware of the particular FPGA. This process is considered graphically with the multiplier tiling methodology, such that the multiplier to be realized is represented as a rectangular $2W_X \times 2W_Y$ sized board which is subdivided into a grid according to the bits of the input vectors X, Y, where each position corresponds to a partial product bit of an AND-array-multiplier.

Fig. 1 shows an example of a 6×4 multiplier, assembled from four 3×2 sub-multiplier tiles $M_{1,..,4}$. Each of the 3×2-tiles generates a 5-bit output vector which subsequently has to by processed on the compressor tree to calculate the output vector of the multiplier as shown in Fig. 1b.

TABLE I: Properties of LUT- and DSP-based multipliers [14] targeting AMD FPGAs, where $\text{cost}_e^{\text{mult}}$ and $\text{cost}_t^{\text{tile}}$ are the number of LUT6 for the single multiplier and multiplier+compression, respectively

Type	Shape	A_t	$\text{cost}_t^{\text{mult}}$	$\text{cost}_t^{\text{tile}}$	W_{out}	E_t	D^t
d	1×1	1	1	1.65	1	0.625	0
c	1×2 / 2×1	2	1	2.3	2	0.87	0
b	2×3 / 3×2	6	3	6.25	5	0.96	0
a	3×3	9	5	8.9	6	1.011	0
e	$2\times k$ / $k\times 2$	$2k$	k+1	$1.65k$ + 2.3	k+2	$\frac{2k}{1.65k+2.3}$	0
-	24×17 / 24×17	408	0	26.65	41	15.30	1



(a) 3×3
(b) $3 \times 2/2 \times 3$
(c) $1 \times 2/2 \times 1$
(d) 1×1
(e) $k \times 2$ / $2 \times k$

Fig. 2: Geometric shapes of rectangular LUT-based tiles [14]

A. *Conventional Rectangular Multiplier Tiles*

The previous multiplier tiling based design methods [4], [14], [16], [35] used, aside from DSP tiles, a set of logic-based sub-multiplier tiles listed in Table I and illustrated in Fig. 2. Those tiles possess several properties relevant for the tiling algorithm. First of all, the tiles have a particular shape defined by the input vectors X and Y. The previously used LUT-based multipliers usually had a rectangular shape. Associated with the shape, the tiles cover a particular number of positions on the board which defines their geometric area A_t. Furthermore, the implementation of the sub-multiplier tiles has certain realization costs in terms of 6-input look-up-tables defined by $\text{cost}_t^{\text{mult}}$. Each tile also generates a W_{out} wide output vector of partial product bits calculated by the individual LUTs. The partial product bits also entail additional costs for the calculation of the final result on the compressor tree, which had been shown in [4] experimentally to be about 0.65 LUTs per bit. So the cost of utilizing a particular tile $\text{cost}_t^{\text{tile}}$ can be described as,

$$\text{cost}_t^{\text{tile}} = \text{cost}_t^{\text{mult}} + \widetilde{\text{cost}}_t^{\text{comp}} \tag{2}$$

where $\widetilde{\text{cost}}_t^{\text{comp}}$ is the compression cost of approximately $0.65 \frac{\text{LUT}}{\text{Bit}} \cdot W_{\text{out}}^t$. This can further be used to derive a measure of efficiency E_t for a particular tile as the quotient of the geometric area covered by the tile divided by its realization cost (Area per LUT)

$$E_t = \frac{A_t}{\text{cost}_t^{\text{tile}}}. \tag{3}$$

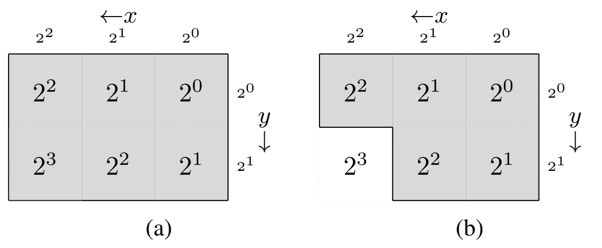


Fig. 3: Motivational example

The efficiency of the tiles is provided for reference purposes to benchmark the properties of the tile, whereas the utilized combined optimization of the tiling and compression uses the exact number of LUTs required for compression.

Tiles like the embedded DSPs as shown in the bottom row of Table I can also possess DSP costs denoted by D^t according to the number of utilized DSPs. Note that the DSP block do not have any LUT cost $\text{cost}_t^{\text{mult}}$ on their own but cause compression costs, so they are listed with $\text{cost}_t^{\text{tile}} = 26.65\text{LUT}$ assuming complete utilization. When underutilized the cost has to be calculated dynamically.

B. Motivational Example

Consider the rectangular 3×2 tile as illustrated in Fig. 3a. The maximal output value in the unsigned case is $(2^3 - 1) \times (2^2 - 1) = 21$, so the required word size is $W_{\text{out}}^t = 5$ bit. Together with the $2+3 = 5$ input bits, the tile is realized using five 5-input LUTs (in the following denoted as 5-LUTs). The 6-LUTs of recent FPGAs can be subdivided into two 5-LUTs, so, three 6-LUTs are necessary to implement this tile. Note that one of the 6-LUTs remains underutilized as it is only occupied by one 5-LUT. The other 5-LUT is typically wasted as all its inputs are already fixed. The overall cost of the tile is

$$\text{cost}_t^{\text{tile}} = \text{cost}_t^{\text{mult}} + 0.65 \cdot W_{\text{out}}^t = 3 + 3.25 = 6.25 \tag{4}$$

and the efficiency with a total geometric area of $A_t = 6$ is

$$E_t = \frac{6}{6.25} = 0.96. \tag{5}$$

The maximal output value can be alternatively computed by summing the weights shown in Fig. 2a, i.e., $2^0 + 2^1 + 2^1 + 2^2 + 2^2 + 2^3 = 21$. Now consider how the properties of the tile would change if the most significant partial product bit with weight 2^3 would be omitted (Fig. 3b). As a consequence, the maximum output value is reduced to $21 - 8 = 13$, thus, requiring only $W_{\text{out}}^t = 4$ output bits. This also leads to a better utilizing of 14 out of 16 possible values (88%) instead of the 22 out of 32 values (69%) of the 3×2 tile.

Accordingly, only four 5-LUTs are needed to realize the tile, which can efficiently be mapped to two 6-LUTs. This corresponds to total realization costs of

$$\text{cost}_t^{\text{tile}} = 2 + 2.6 = 4.6 \tag{6}$$

and an efficiency with the area reduced by one to $A_t = 5$ is

$$E_t = \frac{5}{4.6} = 1.087. \tag{7}$$

It can be observed that the omission of one the partial product bits leads to a new tile variant, which is more efficient then the previously known LUT tiles, and hence allows for tilings which require less LUTs per area. Note that in terms of efficiency the same could be achieved by removing the least significant partial product bit with weight 2^0 at the top right corner, which also necessitates the inclusion of an additional bit for a single position, that can only be mapped as an underutilized LUT. In this case the omitted LSB of the tile is effectively always zero. Without context those structures could appear useless as they do not compute a complete multiplication, but as a tile it can be used to form larger complete multipliers. This observations raise the question which other efficient structures are possible.

III. Searching efficient structures

This section covers the realization of a complete search for efficient structures. As elaborated previously, multiplier tiles can realize random partial products within a larger multiplier, which can be represented by positions on the board. This defines the applicable design space to be explored. Application specific multipliers can be required in random sizes, but the search space has to be constructed to manageable dimensions. The search space has to incorporate all viable combinations that could be mapped within the 6-input LUTs present on most modern FPGAs. The largest rectangular multiplier without the use of the fast-carry-chain is the 3×3, which is constrained by the six available inputs of the LUTs on which the equation of a particular partial product bit may depend on. Hence for example, a 4×2 or 5×1 multiplier would also be possible but is less efficient in terms of covered area per LUT. Also geometrically stretched out versions of the multipliers with gaps between the utilized input variables should be possible within the search space, as they can be realized with otherwise the same properties. This defines the evaluated search space, as combinations beyond this would constitute redundant, even more stretched out versions of other tiles. A geometric bound of 4×4 was used here, which results in 2^{16} possible combinations to be evaluated.

A. Implementation of the Search

To enumerate the evaluated 4×4 search space, as each partial product on the board can either be present or absent, the area was abstracted as a 16-bit binary number. As this includes a considerable number of redundant patterns, each combination not geometrically aligned with active partial products on the top and right of the board are discarded. The evaluation of the patterns is well parallelizable, as it only depends on the respective pattern. So multiple value ranges can be investigated in concurrent tasks. Based on the partial product pattern and the signedness of the input vectors the output bits of the tile can be tabulated as a function of all possible binary combinations of the inputs. From the truth

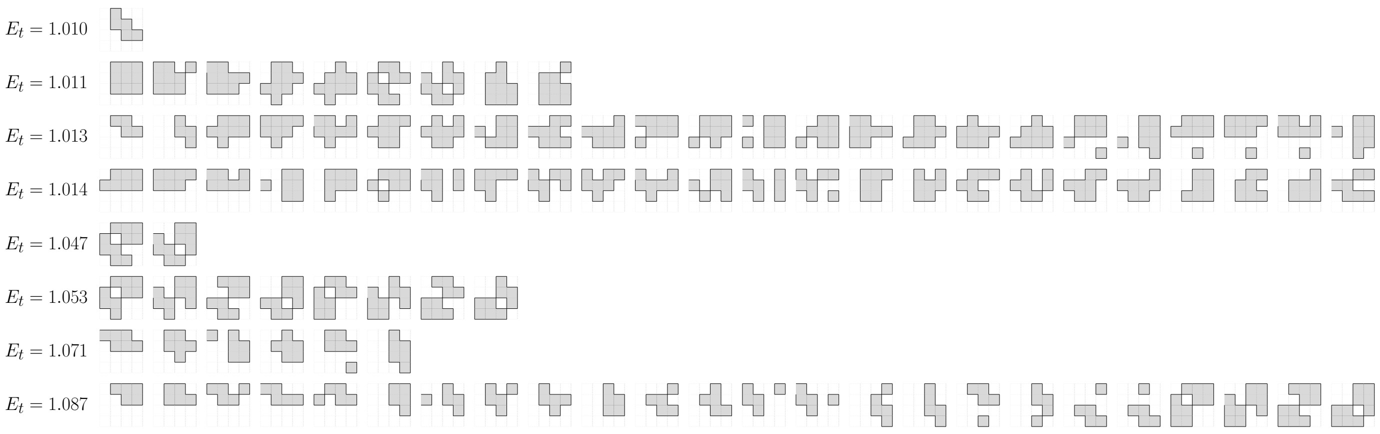


Fig. 4: First 24 Tiles from the efficiency classes above E_t =1.0 of the 4×4 search space

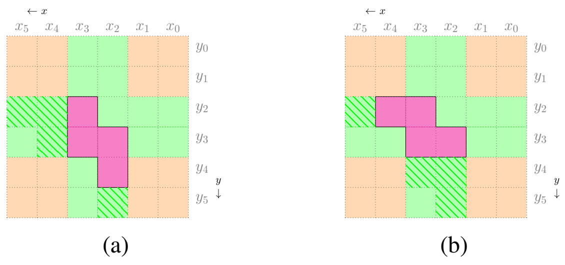

$\leftarrow x$
x_5 x_4 x_3 x_2 x_1 x_0
y_0 y_1 y_2 y_3 y_4 y_5
$y \downarrow$
(a)
(b)

Fig. 5: Design space for Tiles of the highest efficiency

table a set of equations for the tile outputs can be created. The equations were subsequently simplified with the Quine-McClusky method to remove redundant input-dependencies.

The necessity for the simplification is best explained by using an example. Take the truth table of the 3×3 tile in Fig. 2a: it depends on six input bits and produces $3 + 3 = 6$ output bits. Hence, a naive implementation would required six 6-LUTs. However, the simplified equations for the two output LSBs results in

$$r_0 = x_0 y_0 \tag{8}$$

$$r_1 = x_1 y_0 + x_0 y_1 \ . \tag{9}$$

Hence, they only depend on four variables that can be mapped into a single 6-LUT (realizing two 5-LUTs).

Within the resulting equations the number of utilized inputs can be counted to determine the optimal LUT-mapping by evaluating the possible combinations to subdivide a 6-LUT into two 5-LUTs, or placing an equation with 7 input dependencies into two 6-LUTs, etc. So the efficiency of a particular tile in terms of the utilized LUTs in relation the the covered area can be determined to insert the tile to a linked list to hold the results according to its efficiency level.

Note that the search has to be repeated for the four combinations of the signedness of the input vectors X, Y, which in each case can lead to yielding different efficiencies for a particular pattern.

B. *Identified Structures*

The search yielded structures with a multitude of efficiency levels up to $E = 1.087$. Examples of tiles from the efficiency classes above 1.0 are given in Fig. 4. As the tiles with the highest efficiency by themselves are sufficient to generate complete tilings, we focus on those variants that generate overall efficient designs. In total 31 structures were found within the highest efficiency class. At first glance these might appear to contain some randomness (see bottom row of Fig. 4), their structure actually follows a few simple geometric design patterns. As shown in Fig. 5, the tiles can be broken down into in its most generalized form into a horizontal or vertical s-shaped element supplemented by an associated 1×1 element which can be placed in the green area, such that an additional independent partial product bit is generated. All tiles within the highest efficiency class can be derived according to this rule, where some are redundant in the sense that the considered structure is placed twice within a tile. In terms of its properties the tiles cover 5 positions on the board, generate 4 partial product bits and hence require 2 6-LUTs, each subdivided in two 5-LUTs to be realized. For example, consider the s-shaped element in Fig. 5b, which after simplification is described by

$$\begin{aligned} r_0 &= \overline{y_3} y_2 x_3 + y_2 \overline{x_2} x_3 + y_3 \overline{y_2} x_2 + y_3 x_2 \overline{x_3} \\ r_1 &= y_3 \overline{x_2}\,\overline{x_4} x_3 + y_3 x_2 x_4 x_3 + \overline{y_3} y_2 x_4 + y_2 x_4 \overline{x_3} + y_3 \overline{y_2} x_3 \\ r_2 &= y_3 y_2 x_4 x_3 + y_3 y_2 x_2 x_3 . \end{aligned}$$

Outputs r_1, r_2 already use 5 input variables, but r_0 requires only 4 inputs. Hence, another arbitrary input can be combined with one of these 4 inputs to produce another partial product on a 1×1 tile. So for example when placing the 1×1-element at (3,5), the partial product

$$r_3 = y_5 x_3$$

can be computed without additional cost. Note that in the green hatched area the 1×1-element has to be tabulated separately to permit to map it into two 6-LUTs as the corresponding equation would otherwise depend on 6 variables.

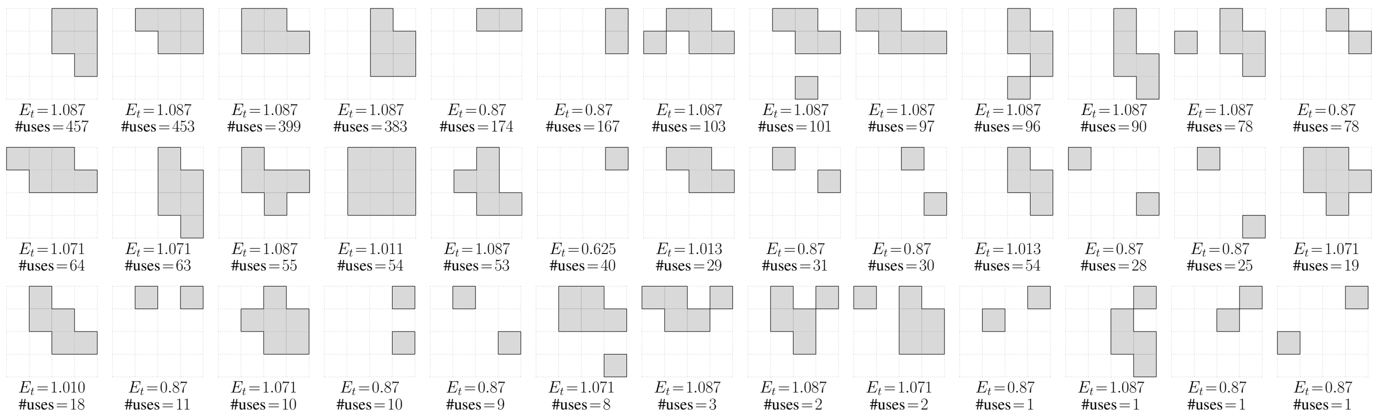


Fig. 6: Geometric shapes of incomplete LUT-based tile-set used for experiments

C. *Selecting the Tile Set for the Optimization*

As the the complete list of tiles from the 4×4 search space yielded an unpractical number of different tiles to be included in the optimization of the tiling, the tile set has to be narrowed down to a selection of the most efficient tile variants. Accordingly, all tiles with an efficiency above 1.0 were selected and cleaned for redundancies which could be described as a combination of different tiles, yielding a tile set of 339 elements. Optimization experiments for all multiplier dimensions between 1×1 and 16×16 were conducted, for which the solver generated optimal solutions up to about 13×14 multipliers within a timeout of 10 h. Beyond this, the solutions were only feasible due to the complexity of the model due to the amount of tiles. Within those experiments 72 of the 339 tiles were actually used. The experiments were subsequently repeated with the set of 72 used tiles, of which 54 were actually used in the next round. Now all the results were optimal within the timeout and improvements could be observed in the cases that were previously not proven to be optimal due to the timeout. When examining those 54 tiles geometrically it became apparent, that repeating redundant elements are present in tiles from some of the efficiency classes, which could be substituted by another tile with higher efficiency and a *helper tile*, while maintaining the same realization costs, as illustrated in Fig. 7.

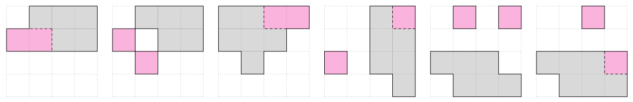


Fig. 7: Examples of the decomposition of tiles in combinations of other helper tiles

It can be observed, that the introduction of the helper tiles does not only allow for the substitution of tiles from some efficiency classes in the remaining list, but also the creation of all tiles in the efficiency classes where the redundancies occur from the initial tile set ($E_t = 1.013$ and $E_t = 1.014$ in Fig. 4). Most of the helper tiles cover two positions which are positioned relative to each other within the 4×4 space, which resulted in a tile set of 55 elements. The experiments showed slight improvements over the tile set without substitutions, and that 41 of the 55 tiles were actually used. Those 41 tiles minus the 2×3/3×2 elements which had been introduced as a helper tile, but could be removed without changes to the optimization objective, form the basis for the final tile set that was used for the synthesis experiments. The final tile set is shown in Fig. 6, sorted by their number of uses (#uses) in the experiment.

IV. Results

A. *Experimental Setup*

The proposed incomplete tiles were added to the multiplier tiling framework of the open-source VHDL-code generator FloPoCo [37]. For the experiments, the combined ILP-based optimization of the tiling and the compressor tree of [35] was used in conjunction with Gurobi 11 as the solver. The generated VHDL designs were subsequently synthesized and implemented with Vivado version 2022.1 for a Kintex 7 (xc7k70tfbv484-3) target FPGA.

B. *Comparison With Previous Tiling*

Table II shows a comparison of synthesis experiment results for multipliers up to 16×16 for the proposed incomplete multiplier tiles, as well as the relative improvement in comparison to the tiling using rectangular tiles [35]. Using incomplete multiplier tiles offers improvements in terms of utilized LUTs in the vast majority of cases. The reductions are up to 17.6% with an average of 3.7%. The relative LUT reduction drops with increasing multiplier sizes but is still significant up to 16×16. Hence, incomplete multiplier tiles are most efficient for small multiplier sizes. Some exceptions show no reductions, which are the very small (single LUT per output bit) multipliers and the ones which can be realized efficiently by the efficient 2×k-tiles only. For reference, the average improvement in the considered cases to the inferred multiplier (VHDL "*" operator) is 18.6%.

The objective usually closely matches the synthesis results, and as expected, are in every case better or equal than with the reference tile set. Although there is always some synthesis

TABLE II: LUT results from synthesis for the proposed incomplete multiplier tiling, as well as the rel. improvement in comparison to previous tiling [35] (in brackets) for $W_X \times W_Y$ multipliers

$W_Y \backslash W_X$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	**1 (0%)**	1 (0%)	2 (0%)	2 (0%)	3 (0%)	3 (0%)	4 (0%)	4 (0%)	5 (0%)	5 (0%)	6 (0%)	6 (0%)	7 (0%)	7 (0%)	8 (0%)	8 (0%)
2	1 (0%)	**2 (0%)**	3 (0%)	5 (0%)	6 (0%)	7 (0%)	8 (0%)	9 (0%)	10 (0%)	11 (0%)	12 (0%)	13 (0%)	14 (0%)	15 (0%)	16 (0%)	17 (0%)
3	2 (0%)	3 (0%)	**5 (0%)**	9 (10.0%)	12 (0%)	14 (6.6%)	16 (5.8%)	18 (5.2%)	21 (0%)	23 (4.1%)	25 (7.4%)	28 (0%)	30 (3.2%)	32 (3.0%)	34 (2.8%)	35 (7.8%)
4	2 (0%)	5 (0%)	9 (10.0%)	**12 (7.6%)**	15 (11.7%)	18 (5.2%)	20 (9.0%)	23 (11.5%)	26 (10.3%)	28 (12.5%)	31 (11.4%)	34 (10.5%)	37 (9.7%)	39 (11.3%)	42 (10.6%)	45 (10.0%)
5	3 (0%)	6 (0%)	12 (0%)	14 (17.6%)	**18 (10.0%)**	23 (-4.5%)	27 (3.5%)	30 (3.2%)	35 (0%)	38 (2.5%)	41 (2.3%)	46 (3.8%)	49 ((0%)	52 (1.8%)	56 (0%)	60 (0%)
6	3 (0%)	7 (0%)	14 (6.6%)	18 (-5.8%)	22 (0%)	**27 (0%)**	34 (0%)	38 (0%)	42 (0%)	46 (0%)	50 (0%)	54 (0%)	58 (0%)	62 (0%)	66 (0%)	70 (0%)
7	4 (0%)	8 (0%)	16 (5.8%)	20 (9.0%)	27 (3.5%)	34 (0%)	**37 (9.7%)**	42 (10.6%)	47 (7.8%)	52 (8.7%)	58 (6.4%)	63 (7.3%)	68 (8.1%)	73 (7.5%)	78 (8.2%)	83 (6.7%)
8	4 (0%)	9 (0%)	17 (10.5%)	23 (11.5%)	30 (3.2%)	38 (0%)	42 (10.6%)	**48 (7.6%)**	54 (8.4%)	61 (6.1%)	67 (5.6%)	73 (5.1%)	79 (3.6%)	85 (4.4%)	91 (5.2%)	97 (3.0%)
9	5 (0%)	10 (0%)	21 (0%)	26 (10.3%)	36 (-2.8%)	42 (0%)	47 (6.0%)	54 (10.0%)	**61 (7.5%)**	69 (5.4%)	75 (6.2%)	81 (5.8%)	88 (4.3%)	96 (4.0%)	101 (4.7%)	108 (5.2%)
10	5 (0%)	11 (0%)	23 (4.1%)	28 (12.5%)	38 (2.5%)	46 (0%)	52 (10.3%)	61 (6.1%)	68 (6.8%)	**75 (8.5%)**	84 (5.6%)	91 (5.2%)	99 (3.8%)	106 (3.6%)	115 (1.7%)	121 (2.4%)
11	6 (0%)	12 (0%)	25 (3.8%)	31 (11.4%)	41 (2.3%)	50 (0%)	58 (6.4%)	68 (4.2%)	76 (5.0%)	83 (6.7%)	**93 (7.9%)**	102 (6.4%)	109 (7.6%)	117 (7.1%)	126 (5.2%)	134 (5.6%)
12	6 (0%)	13 (0%)	26 (7.1%)	34 (10.5%)	47 (-2.1%)	54 (0%)	65 (5.7%)	73 (6.4%)	81 (5.8%)	91 (5.2%)	102 (5.5%)	**110 (6.7%)**	120 (4.0%)	128 (3.7%)	137 (4.1%)	146 (4.5%)
13	7 (0%)	14 (0%)	30 (3.2%)	37 (9.7%)	49 (2.0%)	58 (0%)	68 (8.1%)	79 (3.6%)	88 (5.3%)	98 (4.8%)	109 (6.8%)	120 (6.2%)	**129 (6.5%)**	140 (4.1%)	149 (5.0%)	160 (3.6%)
14	7 (0%)	15 (0%)	32 (3.0%)	39 (11.3%)	52 (1.8%)	62 (0%)	73 (8.7%)	85 (4.4%)	94 (6.0%)	106 (3.6%)	118 (4.8%)	128 (5.1%)	139 (4.7%)	**153 (3.1%)**	162 (3.5%)	173 (2.8%)
15	8 (0%)	16 (0%)	34 (2.8%)	42 (10.6%)	56 (1.7%)	66 (0%)	78 (7.1%)	92 (3.1%)	101 (4.7%)	114 (2.5%)	126 (5.9%)	137 (4.1%)	149 (5.0%)	163 (2.9%)	**175 (5.4%)**	186 (3.6%)
16	8 (0%)	17 (0%)	35 (7.8%)	45 (10.0%)	60 (0%)	70 (0%)	84 (6.6%)	97 (3.0%)	108 (4.4%)	121 (2.4%)	135 (4.2%)	146 (3.9%)	159 (4.7%)	174 (1.6%)	186 (4.1%)	**199 (2.9%)**

TABLE III: LUT comparison to previous rectangular tiling, fractal synthesis ported to AMD and Booth array

Size	**proposed tiling** [LUT]	**rectangular tiling [35]** [LUT]	**fractal synthesis [11]** [LUT]	**Booth-Array [3]** [LUT]
3×3	5	5	6	9
4×4	12	13	n.a.	17
5×5	18	20	20	23
6×6	27	27	32	35
7×7	37	41	42	39
8×8	48	52	n.a.	51

noise, like in the 4×6-case where the reference result requires one LUT less, which results in a degradation of 5.8%.

Except large aspect ratio cases, the designs mostly consist of incomplete multiplier tiles although the $2 \times k$-multiplier-tile eventually becomes asymptotically more efficient towards $E_t = 1.212$ with rising size and surpasses the efficiency of the incomplete tiles beyond 2×12. The own realization costs per area of the proposed tiles is lower then of the $2 \times k$, but they entail higher compression costs. For example, the 13×13 multiplier in Fig. 8a,c with incomplete tiles achieves overall lower realization cost of 130 LUT (objective) due to the less expensive tiling (70 LUT), even with higher compression costs (60 LUT), compared to the multiplier with rectangular tiles in Fig. 8b,d with 91 LUT (tiling) + 48 LUT (compression) = 139 LUT in total. Nevertheless, the proposed tiles are also used in larger designs, where it is beneficial for the solver to substitute $2 \times k$-tiles for smaller tiles to overall generate less expensive designs for example by strategically rearranging the tiling to minimize the final adder length or effectively utilizing compressor inputs which would otherwise remain unused.

C. Comparison to Various Small State-of-the-Art Multipliers

Table III shows a comparison between the proposed tiling using incomplete multipliers, the conventional tiling with rectangular tiles [35], a Booth-Array-Multiplier [3] and a multiplier optimized with the fractal synthesis approach [10], [11]. Although the fractal synthesis approach was initially devised for the Intel architecture, it was ported to the AMD platform. Note that it does not map efficiently to AMD devices, as an ALM (Intel 6-LUT) contains two full-adders, while

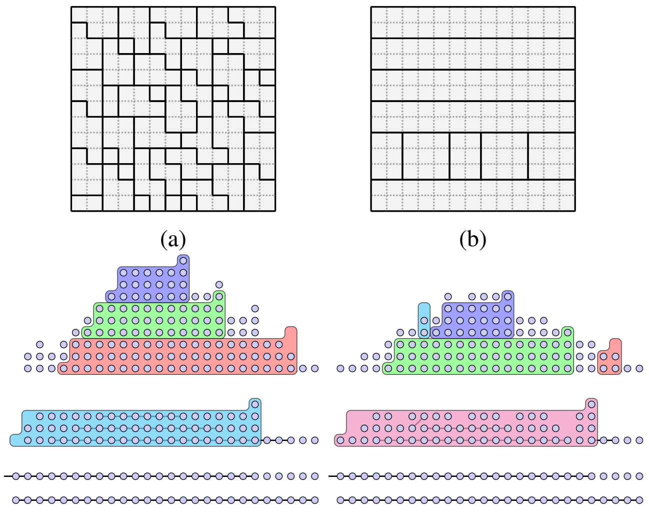


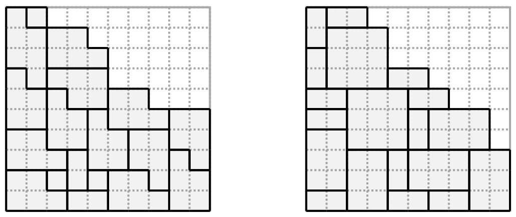
Fig. 8: Tiling and compression of a 13×13-multiplier with (a),(c) incomplete and (b),(d) rectangular sub-multipliers

AMDs 6-LUT contains only logic for one full-adder, resulting in almost twice as many LUTs on AMD. Hence, in our AMD port, we optimized the resources by using ternary addition for the compression. The synthesis experiments were performed for unsigned multipliers. It can be observed that for the examined small multipliers LUT savings can be achieved in many cases. An average improvement to the previous tiling of 7.0%, to fractal synthesis of 13.5% and to the Booth array of 21.6% could be achieved.

D. Comparison with Previous Truncated Tiling

For truncated multipliers, many LSB positions of the partial products do not need to be computed, resulting in tilings where the top-right corner of the board does not need to be covered (see Fig. 9) [16], [38], [39]. Here, the incomplete tiles also prove to be useful as their shape aligns well with staggered pattern between the realized MSBs and omitted LSBs.

Table IV shows the LUT results from synthesis for different sizes up to 16×16 where the output word size W_{out} was set to the input word sizes $W_X = W_Y$. Aside from the degradations of a single LUT due to synthesis noise in the



(a) (b)

Fig. 9: Tiling of a truncated 10×10-multiplier with (a) incomplete and (b) rectangular sub-multipliers

TABLE IV: LUT results for truncated multipliers for $W = W_y = W_y = W_{\text{out}}$

W	**proposed tiling**	**rectangular tiling** [35]	**improvement**
1	1	1	0%
2	1	1	0%
3	3	7	57.1%
4	10	9	-11.1%
5	15	17	11.7%
6	22	22	0%
7	28	32	12.5%
8	37	40	7.5%
9	48	51	5.8%
10	57	67	14.9%
11	68	78	12.8%
12	80	89	10.1%
13	96	104	7.6%
14	111	121	8.2%
15	127	130	2.3%
16	144	148	2.7%

2×2 and 4×4 case, the proposed tiles offer more consistent improvements up to dimensions where the tilings become more similar to the solutions achieved with previous tile set, as the $2\times k$-tiles become more efficient, and thus covers a larger share of the tiled area. Fig. 9 shows a tiling of a faithfully-rounded truncated multiplier [16] with (a) incomplete and (b) rectangular sub-multipliers. While the tiling with rectangular tiles in Fig. 9b requires 37 6-LUTs, only 31 6-LUT suffice for the tiling of Fig. 9a, with compression costs of 28 and 29 6-LUT, respectively (objective).

E. Packing Experiment

To assess how well the proposed structures scale when used in more complex circuits, packing experiments were performed. Aside from LUT resources, the packing efficiency also considers the strain on routing resources which are harder to predict, as there are no detailed reports of the synthesis tools [10], [11]. Following the methodology of previous work, the target device is iteratively filled with instances of the considered multiplier until place and route fails due to a lack of resources. The experiments were conducted for a combinatorial and pipelined (1 stage) unsigned 7×7 multiplier for the proposed method, the multiplier tiling with rectangular tiles [35], the Vivado mult_gen IP-core v12 [40], the inferred multiplier (VHDL "*" operator), the Booth-Array [3], [7], and a multiplier optimized with the fractal synthesis approach [11], ported to the AMD architecture. The experiments for the critical path delay (CPD) were performed with implementations of the tested multipliers sandwiched between registers, to ensure realistic conditions, similar as they would be when integrated in a larger design. Predictions for the theoretical number of instances which should fit into the FPGA were performed based on the synthesis results of a single instance, to evaluate if discrepancies arise compared to the actual number of placed instances the due to properties of the designs which are preventing fitment.

TABLE V: Packing experiment results for a 7×7 multiplier with various implementations

	Type	**single mult.**		**#mult/FPGA**		**Utilization [%]**	
		#LUTs	**CPD [ns]**	**theory**	**actual**	**Slice**	**LUT**
combinatorial	Proposed Tiling	36	4.5	1138	1024	100.0	89.9
	Rectangular Tiling [35]	42	4.8	976	964	100.0	97.6
	Xilinx IP speed opt. v12	58	3.7	706	559	98.6	79.0
	Xilinx IP area opt. v12	79	4.2	518	471	100.0	90.8
	Inferred multiplier	51	3.8	803	713	99.9	87.9
	Booth-Array [3]	39	3.9	1051	841	98.5	80.0
	Booth-Array [7]	32	3.8	1281	921	98.8	71.9
	Fractal Synthesis [11]	38	3.7	1078	822	99.6	76.1
pipelined	Proposed Tiling	36	3.0	1138	1024	100.0	89.9
	Rectangular Tiling [35]	41	3.2	1000	773	100.0	77.3
	Xilinx IP speed opt. v12	58	3.3	706	560	100.0	79.2
	Xilinx IP area opt. v12	77	3.2	532	422	99.9	79.3
	Inferred multiplier	51	3.6	803	713	100.0	87.8
	Booth-Array [7]	32	2.6	1281	919	98.9	71.7
	Fractal Synthesis [11]	38	2.5	1078	819	99.4	75.9

The results in Table V show, that the proposed method allows to fit the most instances to the device, although it has among the highest packing density within the slices. It can be observed that none of the evaluated methods possess properties that would prevent the utilization of vast majority of the slices. Unexpectedly, the Vivado IP core generator with the "speed" goal produces more compact multipliers than with the "area" goal, which confirms the finding in [3].

The packing result of [11] is worse than expected from their experiment which might be due to our AMD-port and modifications. It can generally be observed, that implementations that heavily rely on the fast-carry-chain achieve a lower LUT utilization, as residual LUTs within a slice can not be integrated in a different carry-chain. As expected, the proposed tiling has a lower LUT utilization (89.9%) compared to the rectangular tiling (97.6%) due to the higher use of LUT inputs and outputs. However, the low LUT usage per multiplier still dominates and leads to a significant increase of multiplier instances on the device.

V. Conclusion

The results show, that using incomplete sub-multipliers in the design of multipliers provides advantages in terms of resource utilization in comparison to previous methods. They are especially beneficial for small multipliers as required for current AI inference tasks. Although the experiments were performed for the AMD platform, the idea is applicable to all modern FPGAs. It is demonstrated, that the incomplete tiles can be used effectively with the multiplier tiling methodology to form complete multipliers using the combined

ILP optimization approach for partial product generation and compressor tree design. As shown by the packing experiment, the designs can be used in complex circuits without particular limitations, so the advantages can directly translate to a higher overall arithmetic density in application like AI inference and DSP tasks.

VI. Acknowledgements

The authors would like to thank Killian Klug for the help in implementing and porting previous work on multipliers.

References

[1] 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.

[2] 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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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

[9] 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.

[10] 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

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

[12] 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.

[13] 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.

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

[15] Xilinx Inc., "Whitepaper: Convolutional Neural Network with INT4 Optimization on Xilinx Devices," *WP521 (v1.0.1) June 24, 2020*, 2020.

[16] A. Boettcher, M. Kumm, and F. de Dinechin, "Resource optimal truncated multipliers for FPGAs," in *IEEE Symposium on Computer Arithmetic (ARITH)*, 2021.

[17] A. Meo, "Arithmetic networks and their minimization using a new line of elementary units," *IEEE Transactions on Computers*, vol. C-24, no. 3, pp. 258–280, 1975.

[18] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Efficient synthesis of compressor trees on FPGAs," in *Asia and South Pacific Design Automation Conference*, 2008, pp. 138–143.

[19] H. Parandeh-Afshar, P. Brisk, and P. Ienne, "Improving synthesis of compressor trees on FPGAs via integer linear programming," in *2008 Design, Automation and Test in Europe*, 2008, pp. 1256–1261.

[20] H. Parandeh-Afshar, Brisk, Philip, and P. Ienne, "Exploiting fast carry-chains of FPGAs for designing compressor trees," in *2009 International Conference on Field Programmable Logic and Applications*, 2009, pp. 242–249.

[21] 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

[22] M. Kumm and P. Zipf, "Efficient high speed compression trees on Xilinx FPGAs," in *MBMV*, 2014.

[23] M. Kumm and P. Zipf, "Pipelined compressor tree optimization using integer linear programming," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2014, pp. 1–8.

[24] T. B. Preusser, "Generic and universal parallel matrix summation with a flexible compression goal for Xilinx FPGAs," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–7.

[25] Y. Yuan, L. Tu, K. Huang, X. Zhang, T. Zhang, D. Chen, and Z. Wang, "Area optimized synthesis of compressor trees on xilinx fpgas using generalized parallel counters," *IEEE Access*, vol. 7, pp. 134 815–134 827, 2019.

[26] J. Hormigo, M. Ortiz, F. Quiles, F. J. Jaime, J. Villalba, and E. L. Zapata, "Efficient implementation of carry-save adders in FPGAs," in *2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*, 2009, pp. 207–210.

[27] W. Kamp, A. Bainbridge-Smith, and M. Hayes, "Efficient implementation of fast redundant number adders for long word-lengths in FPGAs," in *2009 International Conference on Field-Programmable Technology*, 2009, pp. 239–246.

[28] J. M. Simkins and B. D. Philofsky, "Structures and Methods for Implementing Ternary Adders/Subtractors in Programmable Logic Devices," *US Patent No 7274211, Xilinx Inc.*, Sep. 2007.

[29] G. Baeckler, M. Langhammer, J. Schleicher, and R. Yuan, "Logic Cell Supporting Addition of Three Binary Words," *US Patent No 7565388, Altera Coop.*, 2009.

[30] L. Dadda, "Some schemes for parallel multipliers," in *Alta Frequenza*, vol. vol. 34, 1965, pp. pp. 349–356.

[31] K. A. C. Bickerstaff, E. J. Swartzlander, and M. Schulte, "Reduced Area Multipliers," in *Application-Specific Array Processors, 1993. Proceedings., International Conference on*. Application-Specific Array Processors, 1993 Proceedings, International Conference on, 00 1993, pp. 478 – 489.

[32] T. Matsunaga, S. Kimura, and Y. Matsunaga, "Power and delay aware synthesis of multi-operand adders targeting LUT-based FPGAs," in *IEEE/ACM International Symposium on Low Power Electronics and Design*, 2011, pp. 217–222.

[33] T. Matsunaga, Kimura, Shinji, and Matsunaga, Yusuke, "An exact approach for GPC-based compressor tree synthesis," *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences*, vol. E96A, pp. 2553–2560, 12 2013.

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

[35] A. Böttcher and M. Kumm, "Towards globally optimal design of multipliers for fpgas," *IEEE Transactions on Computers*, vol. 72, no. 5, pp. 1261–1273, 2023.

[36] 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.

[37] F. de Dinechin and M. Kumm, *Application-Specific Arithmetic - Computing Just Right for the Reconfigurable Computer and the Dark Silicon Era*. Springer, 2024. [Online]. Available: https://link.springer.com/book/9783031428074

[38] 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.

[39] 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.

[40] Xilinx Inc., "Multiplier v12.0, LogiCORE IP Product Guide," *PG108 November 18, 2015*, 2015.