Towards Globally Optimal Design of Multipliers for FPGAs

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

Abstract—The design of a multiplier typically consists of three steps: (1) partial product generation, (2) compressor tree design and (3) the selection of the final adder. Conventionally, these three steps are performed consecutively. However, when targeting FPGAs, there are many possibilities in all three design steps that heavily influence each other. This proposal presents for the first time a holistic optimization, combining all three optimization steps yielding a minimum amount of look-up-tables (LUTs) while it can also guarantee the minimal number of (pipeline) stages. An ILP-formulation for the determination of a combined, globally optimal solution for the multiplier tiling, compressor tree generation and final adder selection is proposed. With globally optimal we mean that the best solution is found for a given set of sub-multipliers for partial product generation, compressors and final adder.This allows to improve the quality and evaluate the limitations of existing heuristic 3-step approaches. It is shown experimentally for the example of Xilinx FPGAs, that globally optimal solutions can be obtained for multiplier sizes of practical relevance, leading to significant LUT reductions. Additional packing density experiments show that a significantly larger number of multiplier instances can be mapped to the same device.

Index Terms—multiplier, FPGA, multiplier tiling, compressor tree, integer linear programming, computer arithmetic

1 Introduction

Multipliers are one of the most essential building blocks of arithmetic circuits, hence their efficient implementation in terms of cost, delay and latency is of vital interest due to the impact on the properties of the overall design. The demand for multiplications was the motivation to first embed multipliers and nowadays DSP blocks (including pre/post adders) as hard blocks into the fabric of FPGAs. DSPs provide the most efficient multipliers and should be used whenever possible. However, there are two major limitations [1]–[4]: 1) When the word size of the DSP is much smaller or larger than required: either DSP resources are wasted or have to be extended with further DSPs or multipliers based on look-up tables (LUTs) to meet the requirements. 2) The number of DSP is insufficient: efficient LUT-based multipliers are necessary.

Therefore, a design method is necessary which allows to design any size of multiplier and to trade DSPs with LUTs. As shown in Fig. 1a, the design of multipliers is usually performed in three steps:

1) The partial product generation.
2) The design of the compressor tree.
3) The design of the final adder.

For FPGAs, this poses some distinct design challenges as there are many possible FPGA mappings in each of the steps. Hence, each of these steps requires its own combinatorial optimization.

Various previous works address the efficient mapping of multipliers to FPGAs [1]–[13]. LUT-based multipliers are covered in [3], [5], [7], [8], [10], [11]. An efficient LUT-mapping of the Baugh-Wooley-multiplier using the fast-carry-chain of recent FPGAs is demonstrated in [5]. A Booth-Array-Multiplier for Xilinx FPGAs is presented in [3], [7], [8]. Another efficient LUT-mapping scheme for small multipliers was presented with the fractal synthesis approach [10], [11]. The realization of two smaller multipliers from one DSP (and additional LUTs) was demonstrated in [9], [11], [12]. Large multipliers using Karatsuba's algorithm to reduce the number of DSPs were addressed in [1], [13].

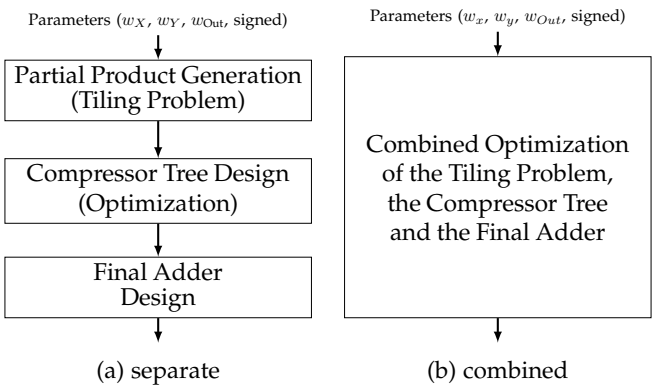


Fig. 1: Traditional (a) and proposed combined (b) multiplier design process.

Any of the previous methods can be used to generate partial products. To design a multiplier with a given arbitrary target multiplier size (1st step in Fig. 1a), the multiplier tiling methodology was devised, primarily to compose multipliers beyond the size of a single DSP block from a minimal number of DSPs [1]. The multiplier tiling methodology has proven to be an effective method to handle the trade-off between DSPs and supplementary LUT-resources to provide resource-efficient solutions for all multiplier sizes [2], [4], [6], [13], [14]. Since then, several heuristics [2], [6], [13]

Funded by the German Research Foundation, Project No. 426369132

and optimal [4], [14] methods were proposed to solve this underlying optimization problem.

The next step is the bit-shifted summation of all the partial products where compressor trees are used in state-of-the-art designs (2nd step in Fig. 1a). On FPGAs, full adders and half adders, the traditional building blocks of compressor trees, do not map efficiently. Instead, generalized parallel counters (GPCs) [15] have been proposed, since they are designed to map well to the given LUT and fast-carry-chain structure of modern FPGAs [16]. Several efficient mappings of GPCs using the 6-input LUTs (LUT6) [16], [17] and the fast-carry-chain [18]–[23] were published. Aside from the GPCs there is also the particularly efficient (4:2) compressor [20], [24], [25]. The introduction of those compressors complicates the compressor tree design considerably, compared to for classic algorithms that only uses full adders and half adders [26], [27]. Multiple heuristic [16], [18], [19] and optimal [17], [21], [28]–[30] solutions were proposed for this.

Ultimately, the final adder has to be designed (3rd step in Fig. 1a). There are basically two options on FPGAs. One is a classic ripple-carry adder (RCA) that features two inputs per column and is using the fast carry-chain. Many modern FPGAs provide the possibility to realize ternary adders, i.e., adders which can add three inputs, by using the same hardware resources as for a common two-input adder [31], [32]. A mix of full adders and ternary adder cells may also be beneficial as demonstrated in [22]. Usually, the designer selects the adder and provides the number of arguments as target size for the compressor tree design.

All these steps have been designed and solved individually. However, they are not independent. First, the partial product selection (the tiling) influences the position of bits in the input stage of the compressor tree, which can heavily influence the cost of the compressor tree. Next, the compressor tree design influences the word size and shape of the final adder, ignoring the fact that a slightly more complex compressor tree can lead to a smaller final adder. Finally, the tiling algorithm can only approximate the actual compression costs, since those will only be known after the completion of the compression algorithm, but has to resort to using statistical average compression costs per bit.

This work presents a holistic optimization, combining all three optimization steps in a globally optimal approach using integer linear programing (ILP) as shown in Fig. 1b. With *globally optimal* we mean that the best solution (least resources with optional minimum compression stages) is found for a given set of sub-multipliers for partial product generation, compressors and final adder. So far, only two of the three optimization steps have been combined in the literature, usually the compressor tree and the final adder in a heuristic [27] or optimal way [33]. A recent optimal approach offers resource advantages for ASICs [33]. In contrast to FPGAs, the partial product generation on ASICs is trivial as AND gates or Booth encoding are considered state-of-the-art that do not require further optimization.

This proposal

- presents for the first time a combined optimization of the partial product generation (tiling), the compressor tree design, and final adder design,
- demonstrates the integration of the signed case and the dynamic calculation of the combined sign extension constant vector,
- demonstrates the design of faithfully rounded truncated multipliers, with the dynamic calculation of the resulting rounding and error re-centering constant,
- extends the ILP-formulation for faithfully rounded truncated multipliers [14] to generate valid results for the signed truncated case.
- evaluates the proposed methodology for the example of Xilinx FPGAs in comparison with the conventional separate optimization as well as multipliers obtained from vendor tools in terms of LUT resources, delay, latency and packing density.

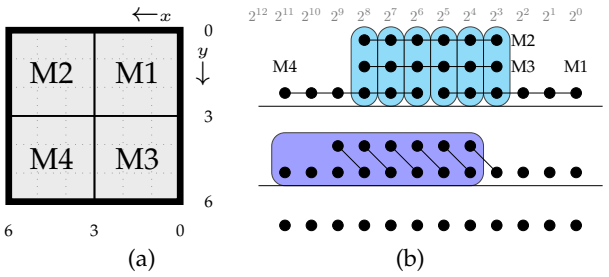


Fig. 2: Tiling (a) and compressor tree (b) of a 6×6-multiplier, composed of four 3×3-tiles.

2 Previous Multiplier Design Process

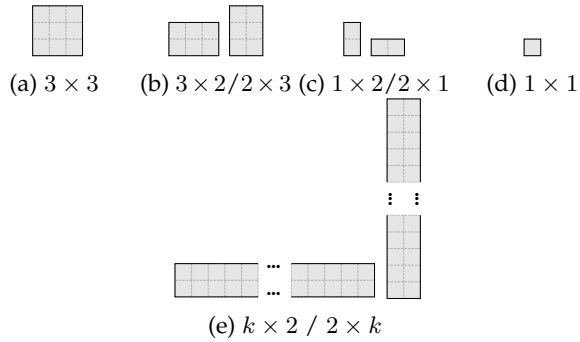
Consider Fig. 2 as an example for the applied multiplier design. Fig. 2a shows the tiling representation of a 6×6 multiplier, represented as a 6×6 board. The multiplier is realized by four 3×3 sub-multipliers, each represented as a 3×3 tile. Those sub-multipliers generate the four 6-bit wide partial products $M_{1,..,4}$, that enter the first stage of the compressor tree, represented as dot diagram in Fig. 2b. The compressor tree summates the partial products in the subsequent stages by compressors (full adders marked blue in the example, compressing three bits in one column and produce two bits in one row) and the final adder (purple) to form a single output vector. The steps are detailed in the following including the previous models to solve each step individually.

2.1 Multiplier Tiling

Multiplier tiling is the process of designing large multipliers from (smaller) sub-multipliers. As shown in Fig. 2a, a large multiplication $X \times Y$ of size $2W \times 2W$ can be subdivided into four smaller sub-multiplications $M_{1,..,4}$, by splitting X and Y into sub-words of size W each

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

Hence, the multiplication can be represented as weighted sum of the partial products of $M_{1,..,4}$. This partitioning of larger multipliers can be performed down to the individual partial product bits. The multiplier to be



(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. 3: Geometric shapes of the LUT-based tiles [13]

TABLE 1: Properties of LUT- and DSP-based multipliers [13] targeting Xilinx 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

Tile type	Shape	Tile area	$\text{cost}_t^{\text{mult}}$	$\text{cost}_t^{\text{tile}}$	w_{Out}	D^t
1	1×1	1	1	1.65	1	0
2/3	1×2 / 2×1	2	1	2.3	2	0
4/5	2×3 / 3×2	6	3	6.25	5	0
6	3×3	9	5	8.9	6	0
7/8	2×k / k×2	$2k$	k+1	k+2	1.65k + 2.3	0
9/10	24×17 / 24×17	408	0	26.65	41	1

designed can be interpreted geometrically as a board with the dimensions $w_X = 2W$ and $w_Y = 2W$ according to the input width of the multiplicands as shown in Fig. 2a. Every position on the board corresponds to a partial product as it would be generated by the AND-gate of a Baugh-Wooley multiplier. However, in most practical tilings larger sub-multipliers that better utilize the FPGA resources are typically more efficient.

The sub-multipliers can be the embedded DSP-blocks of modern FPGAs or LUT-based multipliers. A tiling is considered to be valid when each position on the board is covered by exactly one tile, such that each partial product bit is represented in the output vectors of the tiles. The overlapping of tiles is prevented and considered disadvantageous, since it would require compensation and would unlikely result in the least expensive solution.

2.1.1 Utilized Multiplier Tiles

This work uses a subset of previously published state-of-the-art sub-multipliers designed for Xilinx FPGAs shown in Fig. 3 and Table 1, which turned out to be the most effective, although the methods proposed here are not limited to those tiles. We limit ourself to current Xilinx FPGAs in this work but the presented methods are generic and are portable to other FPGA architectures. The particular properties of those tiles are listed in Table 1 which are explained in the following. Each tile is defined by its shape and the corresponding geometric area covered by the tile on the board. For each tile of type t, the required LUT-costs (as the number of LUT6 equivalents) are denoted by $\text{cost}_t^{\text{mult}}$ and D^t specifies the number of used DSP blocks. For example, the last tile in Table 1 corresponds to the 24×17 (in unsigned case) DSPE48E found in Xilinx devices.

Furthermore, the tiles produce partial products bits w_{Out}, that usually entail additional costs for their compression. Since the exact cost for the compression of the bits can not be known before the completion of the compressor tree design, an estimate $\widetilde{\text{cost}}_t^{\text{comp}}$ is used in the previous tiling ILP-formulations [4], [13], [14]. It was shown experimentally in [4] to be 0.65 LUT per bit with the applied compressor tree design method, leading to a total LUT cost of

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

The multiplier set contains several LUT-based multipliers, like the 3×3 shown in Fig. 3a which can be mapped to five LUT6 [6] and also generates an output vector of 6 bits as partial product. Additionally there are more efficient (in terms of covered area per LUT) 3×2 [3] and 2×1 [4] multipliers, that map five LUT5 in three LUT6 and two LUT5 in one LUT6, respectively. To guarantee valid solutions without overlapping the border of the tiled area, also a 1×1-tile is included. Also included is the FPGA mapping of a Baugh-Wooley multiplier [5] with a size of $2 \times k$, that is particularly efficient for large multipliers since it can already compress two partial products per LUT using the integrated fast-carry-chain.

DSP-based multipliers also provide not only multipliers but also pre-adders and post-adders. Previous work proposed to combine several DSP results using the post-adders, leading to so-called super-tiles [1]. Other work used Karatsuba's method to combine several DSPs using pre-adders and post-adders [34]. These are interesting for relatively large multipliers with >32 bits but are not used in this work.

2.1.2 Previous ILP Tiling Formulation

We will now introduce the ILP multiplier tiling methodology presented in [4] that solves step 1 of Fig. 1a. This work extends it to also solve steps 2+3 in a holistic way. All variables and constants used in this and the following formulations are summarized in Table 2. The optimal tiling provides all the positions and tile types for sub-multipliers such that each field of the board is covered once and the sum of all individual costs ($\text{cost}_t^{\text{tile}}$) is minimal.

For that, a decision variable $d_{x,y}^t$ is defined which is true if (and only if) tile of type t is placed at position (x, y). The objective is to minimize the total cost which is given as the sum of the decision variables weighted by their respective costs $\text{cost}_t^{\text{tile}}$ for every possible placement (x, y) and every possible tile of type t from the available set of tiles

$$\min \sum_{t=0}^{T-1} \sum_{x=0}^{w_X-1} \sum_{y=0}^{w_Y-1} \text{cost}_t^{\text{tile}} d_{x,y}^t \ . \tag{3}$$

To model that each position of the board should be covered exactly once, additional constraints have to be added. The board shape is modeled by a constant $O_{x,y} \in \{0, 1\}$ that defines the shape of the multiplier. It is true for every position to be covered (see light gray area in Fig. 4). This allows to omit particular positions as it is needed for truncation. Similarly, a constant $o_{x,y}^t \in \{0, 1\}$ is defined for each tile, describing its shape (see dark gray area in Fig. 4, representing a 2×3 tile). Its coordinates are relative to its origin and true as long as x and y are within the shape of the tile.

TABLE 2: Variables (top) and constants (bottom) used in ILP formulations

Variable/ Constant	Meaning
$d^t_{x,y} \in \mathbb{N}_0$	Decision variable for tile of type t placed at x, y
$k_{s,e,c} \in \mathbb{N}_0$	Number of compressors e in stage s and column c
$h^t_{x,y} \in \mathbb{N}_0$	Sign extension vector for tile of type t at x, y, if required
$v_c \in \mathbb{N}_0$	Individual bit of the constant vector in column c
$D^t \in \mathbb{N}_0$	DSP Cost (in DSPs) of tile of type t
$\text{cost}^{\text{comp}}_e \in \mathbb{N}_0$	Cost (in LUTs) of compressor e
$\text{cost}^{\text{mult}}_t \in \mathbb{N}_0$	Cost (in LUTs) of tile of type t (just the multiplier)
$\text{cost}^{\text{tile}}_t \in \mathbb{R}_0$	Cost (in LUTs) of tile of type t (including avg. compression)
$N_{s,c} \in \mathbb{N}_0$	Number of bits in stage s and column c
V	Sign extension vector
$w_X \in \mathbb{N}_0$	Width of the multiplier
$w_Y \in \mathbb{N}_0$	Height of the multiplier
$o^t_{x,y} \in \{0,1\}$	Defines positions covered by tile of type t placed at x, y relative to it origin; true as long as x and y are within the shape of the tile
$O_{x,y} \in \{0,1\}$	Defines if position x, y has to be covered
$g^t_{c,x,y} \in \{0,1\}$	Defines if a tile of type t at x, y contributes into column c
$T \in \mathbb{N}_0$	Number of available multiplier tiles
$M_{s,c} \in \mathbb{N}_0$	Number of bits removed by compressor e in column c
$K_{s,c} \in \mathbb{N}_0$	Number of bits generated by compressor e in column c
$E \in \mathbb{N}_0$	Number of available compressing elements
$S \in \mathbb{N}_0$	Number of stages of the compressor tree
$C \in \mathbb{N}_0$	Number of columns of the compressor tree
$C_e \in \mathbb{N}_0$	Number of input columns used by compressor e
$R \in \mathbb{N}_0$	Rounding constant in truncated case

Then, the number of overlapping tiles for a given coordinate (x, y) can be represented as the sum of decision variables, weighted by the tiling shape constant. By constraining the overlap to be one for every possible position

$$\text{C1:} \quad \sum_{s=0}^{T-1} \sum_{x'=0}^{x} \sum_{y'=0}^{y} o^t_{x-x',y-y'}\, d^t_{x',y'} = 1 \,, \quad \left.\begin{array}{l} \text{for } 0 \le x < w_X, \\ 0 \le y < w_Y \\ \text{with } O_{x,y} = 1 \end{array}\right\}$$

we yield a valid tiling solution.

Since the embedded DSP-Blocks are a limited resource it might be advantageous to limit their use to trade between DSPs and LUTs. Accordingly the DSP costs of all available tiles t at every possible position are accumulated in an equation by summating the decision variables $d^t_{x,y}$ weighted with their respective DSP counts D^t and constrained by the desired maximal number

$$\text{C2:} \quad \sum_{t=0}^{T-1} \sum_{x=0}^{w_X-1} \sum_{y=0}^{w_Y-1} D^t d^t_{x,y} \le \#\text{DSP} \,.$$

2.2 Compressor Tree Design

The goal of the compressor tree design method is to find the cheapest combination of compressors from a set of available compressors that compress the partial products (see top of Fig. 2b) to at most two (three) rows, such that its final sum can be computed by a two-input (ternary) adder (see bottom of Fig. 2b). We first present the state-of-the-art compressors for FPGAs and then discuss a previously proposed ILP formulation to solve the compression problem.

2.2.1 Utilized Compressors

The utilized compressors are highly optimized circuits, that expect several bits from one or more columns of the compressor tree as inputs and adds them to calculate output bits that enter several columns of the subsequent stage of the compressor tree, such that the total number of bits is reduced.

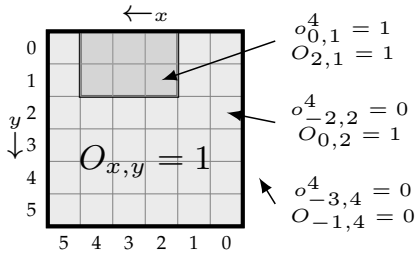


Fig. 4: Example of the coverage constants for a 6×6-multiplier $O_{x,y}$ and a 3×2 tile of type $t = 4$ $o^4_{x,y}$ placed at $(2, 0)$. Coordinates are shown to illustrate that $O_{x,y} = 1$ for for all absolute coordinates (x, y) that have to be covered and that $o^4_{x,y} = 1$ for all coordinates (x, y) which are covered by the tile, relative to its origin $(2, 0)$

In Table 3, the various compressors are summarized and characterized. They can be distinguished between GPCs (upper part of Table 3) and row adders (lower part of Table 3). GPCs are characterized by their input and output sizes, denoted by comma separated input sizes followed by the length of the output vector after the semicolon in the round brackets. For example, GPC $(2, 3; 3)$ takes two bits from column 1, and three bits from column 0 and generates three output bits (the weighted sum of the input bits). An important metric of the compressors is their efficiency, which is defined as the quotient between the difference δ between input bits b_i and output bits b_o (i.e., bits the compressor removes) and the realization cost of the compressor.

$$E = \frac{b_i - b_o}{k} = \frac{\delta}{k} \tag{4}$$

The delay τ of a particular compressor depends on utilized components consisting of LUTs (τ_{L}) and the carry chain (τ_{CC}). As $\tau_{\text{CC}} \ll \tau_{\text{L}}$, short carry chains can be ignored, which simplifies the comparison [30]. The delay and cost of the row adder depends on their size k.

2.2.2 Pipelining

The term *stage* as used within this work refers to the compressor stages of a compressor tree and not to pipeline stages. However, each stage is also a perfect candidate for a pipeline stage as each compressor can be equipped with a pipeline register without cost by activating an (otherwise unused) flip-flop. Costs have only to be considered for flip-flops used to balance the pipeline. To model these, a single-input, single-output compressor (1;1) was added to the set of compressors. In stages that represent a pipeline stage, it can be used to model the flip-flop cost while in other stages it represents a wire without cost. The cost of each flip-flop is considered to be 0.5 as for each LUT6, two flip-flops exist on modern FPGAs [21]. Based on a target clock period T_{clk}, pipeline stages are introduced every $\lfloor T_{\text{clk}}/\tau \rfloor$'th compressor stage.

2.2.3 Previous Compressor Tree Design ILP-Formulation

We will now introduce the compressor tree design ILP formulation presented in [30] that solves step 2 of Fig. 1a and that we will combine with [4] for our holistic optimization.

TABLE 3: High efficient GPCs and adders targeting Xilinx FPGAs (where $\text{cost}_e^{\text{comp}}$ is the number of LUT6, τ_{L} , τ_{CC} and τ_R describe the delay of a LUT, one bit of carry propagation in the carry chain and a local routing delay, respectively, with $\tau \approx \tau_{\text{L}} \approx \tau_R$)

GPC/row adder	Ref.	$\text{cost}_e^{\text{comp}}$	Efficiency ($E = \delta/k$)	Delay
(6;3)	[6]	3	1	$\tau_{\text{L}} \approx \tau$
(5;3)		2	1	$\tau_{\text{L}} \approx \tau$
(1,4;3)		2	1	$\tau_{\text{L}} \approx \tau$
(2,3;3)	[20]	2	1	$\tau_{\text{L}} + 2\tau_{\text{CC}} \approx \tau$
(3;2)	[6]	1	1	$\tau_{\text{L}} \approx \tau$
(1,4,1,5;5)	[20]	4	1.5	$\tau_{\text{L}} + 4\tau_{\text{CC}} \approx \tau$
(1,3,2,5;5)	[21]	4	1.5	$\tau_{\text{L}} + 4\tau_{\text{CC}} \approx \tau$
(2,1,1,7;5)	[23]	4	1.5	$\tau_{\text{L}} + 4\tau_{\text{CC}} \approx \tau$
(6,2,3;5)	[22]	4	1.5	$\tau_{\text{L}} + 4\tau_{\text{CC}} \approx \tau$
(6,1,5;5)	[22]	4	1.75	$\tau_{\text{L}} + 4\tau_{\text{CC}} \approx \tau$
(1,4,0,7;5)	[23]	4	1.75	$\tau_{\text{L}} + 4\tau_{\text{CC}} \approx \tau$
(6,0,7;5)	[23]	4	2	$\tau_{\text{L}} + 4\tau_{\text{CC}} \approx \tau$
2-input add.		k	1	$\tau_{\text{L}} + k\tau_{\text{CC}}$
ternary add.		k	$2 - \frac{2}{k}$	$2\tau_{\text{L}} + \tau_R + k\tau_{\text{CC}}$
4:2 comp. (a/b)	[20]	k	$2 - \frac{2}{k}$	$\tau_{\text{L}} + k\tau_{\text{CC}}$

It is realized as an iterative process where the number of stages is increased until a feasible solution is found. This ensures that the minimal number of stages is found which is advantageous in terms of latency and delay. The solver usually quickly detects if a model is infeasible due to an insufficient number of stages. Variable $N_{s,c}$ models the number of bits of column c in compression stage s of the compressor tree. Hence, the input values for the model are the number of bits $N_{0,c}$ that enters the first stage $s = 0$, as they are generated by the partial products of the sub-multipliers

$$\text{C3}: \qquad N_{0,c} = \text{no. of input bits in column } c.$$

The solution is described by integer variables $k_{s,e,c}$, that count the number of compressors of type e, placed in column c of stage s. For example, assuming the FA in Fig. 2b is compressor element $e = 0$, the solution is represented as $k_{0,0,3\ldots8} = 1$ (all other $k_{s,e,c}$ are zero).

The objective is to minimize the total cost of compressors, which is obtained by the sum of the individual compressor costs $\text{cost}_e^{\text{comp}}$

$$\min \sum_{s=0}^{S-1} \sum_{c=0}^{C-1} \sum_{e=0}^{E-1} \text{cost}_e^{\text{comp}} k_{s,e,c} \ . \tag{5}$$

Now, for every stage s and column c of the compressor tree, one constraint is generated, to describe the number of bits $M_{e,c}$ absorbed by the available compressors $e = 0 \ldots E-1$ placed between the first column $c' = 0$ and the current column, when they are placed according to the integer variable $k_{s,e,c'}$. It is also possible to connect the compressor inputs to zero constants $Z_{s,c} \geq 0$ if an otherwise favorable compressor placement $k_{s,e,c'}$ results in superfluous inputs in a particular column c

$$\text{C4}: \quad N_{s,c} = \sum_{e=0}^{E-1} \sum_{c'=0}^{c} M_{e,c} k_{s,e,c'} - Z_{s,c} \Bigg\} \begin{array}{l} \text{for } 0 \leq s < S, \\ 0 \leq c < C \ . \end{array}$$

Another constraint is used to sum up the number of bits $K_{e,c}$ that enter a particular column c of the subsequent stage s from the outputs of the possible compressors e when they are placed in a column c' relative to the current column c between the current s and the next stage, as described by the individual integer variables $k_{s,e,c'}$

$$\text{C5}: \quad N_{s+1,c} = \sum_{e=0}^{E-1} \sum_{c'=0}^{c} K_{e,c} k_{s,e,c'} \Bigg\} \begin{array}{l} \text{for } 0 \leq s < S, \\ 0 \leq c < C \ . \end{array} \qquad .$$

As it is usually advantageous to place a two-input ripple carry adder or ternary adder as the final compression stage, the number of bits in the columns of the compressor tree c before the final adder $s = S - 1$ are constrained to be 2 or 3 according to the applied adder type, to obtain a output vector with a max. column height of one in the output stage ($s = S$).

$$\text{C6}: \qquad N_{s,c} \leq \begin{cases} 3 & \text{for } s = S-1 \text{ AND ternary adder} \\ 2 & \text{for } s = S-1 \text{ AND } 2\text{−input adder} \end{cases}$$

Since those row adders have a variable number of columns, they are modeled using high $k_{s,e_H,c+1}$ middle $k_{x,e_M,c+1}$ and low $k_{s,e_L,c}$ elements that can only be instantiated in this particular order, their placement is constrained by

$$\text{C7}: k_{s,e_H,c+1} + k_{x,e_M,c+1} = k_{s,e_M,c} + k_{s,e_L,c} \Big\} \begin{array}{l} \text{for } s = S-1, \\ 1 \leq c < C \ . \end{array}$$

3 Proposed Holistic Tiling & Compression

The previous ILP tiling and compression methods provided only locally optimal solutions, while the proposed holistic method combines both models to find globally optimal solutions. We start with describing the combined model and then extend it to consider the final adder, signed and/or truncated cases.

3.1 Combined ILP Formulation

To obtain a globally optimal solution for the multiplier, a combined formulation is proposed. This gives the solver an additional level of freedom to find tilings whose output bits can be most efficiently compressed by the available compressors. In the combined model the tiling constraints C1 and C2 remain unchanged. While in the independent optimizations the output bits from the tiling were used as constants in constraint C3 for the compressor tree design, they become variables in the combined model. To couple both formulations, the decision variable $d_{x,y}^t$ of every available tile $t = 0 \ldots T-1$ at every possible position $0 \leq x < w_X$ and $0 \leq y < w_Y$ that can contribute to a particular column $c = 0 \ldots C-1$ of the input stage of the compressor tree ($s = 0$) according to its contribution constant $g_{c,x,y}^t$ for a tile of type t placed at position (x, y), is added to the new C3′ constraint for the corresponding column c, replacing C3 by

$$\text{C3}': \quad N_{0,c} = \sum_{t=0}^{T-1} \sum_{x=0}^{w_X-1} \sum_{y=0}^{w_Y-1} g_{c,x,y}^t d_{x,y}^t \Bigg\} \text{ for } 0 \leq c < C \ .$$

Note that some sub-multipliers might be pipelined as well to provide the full performance like, e. g., DSP blocks.

Instead of adding them in the input stage $s = 0$, their outputs can be considered in the compressor tree stage that corresponds to its pipeline stage by adding those bits to constraint C4 in a similar way as done in C3′. However, due to space consideration, we will describe only the simple case here.

Furthermore, the objective functions have to be combined, such that the sum of the tiling and compression cost is determined. In contrast to the tiling formulation, where the cost $\text{cost}_t^{\text{tile}} = \text{cost}_t^{\text{mult}} + \widetilde{\text{cost}}_t^{\text{comp}}$ of a particular tile type t were considered including the average compression costs for the generated number of output bits ($\widetilde{\text{cost}}_t^{\text{comp}}$), in the combined optimization, only the realization costs for the tile ($\text{cost}_t^{\text{mult}}$) have to be considered, since the precise compression costs are already considered in the compression part of the cost function. This leads to the objective

$$\min \sum_{t=0}^{T-1} \sum_{x=0}^{w_X-1} \sum_{y=0}^{w_Y-1} \text{cost}_t^{\text{mult}} d_{x,y}^t + \sum_{s=0}^{S-1} \sum_{c=0}^{C-1} \sum_{e=0}^{E-1} \text{cost}_e^{\text{comp}} k_{s,e,c}. \tag{6}$$

Note that this formulation by itself will find the solution with the least possibly number of stages. To find the solution with the least cost, it has to be called iteratively with higher stage counts until the cost rises again.

3.2 Consideration of the Final Adder

A further optimization is the consideration of the final adder during the compressor tree design. Constraint C6 only requires the maximal column height in the penultimate stage to be less or equal to the input height of the final adder, so that the realization of the final adder is performed as a post-processing step. In contrast to this, in the revised formulation the maximal column height in output stage $s = S$ is constrained to one and the final adder is considered in the cost function among the other compressors. This allows the ILP formulation to also minimize the length of the final adder, by influencing the tiling and the compressors in previous stages, when this offers a cost advantage. This is done by replacing C6 with

$$\text{C6}' : \qquad N_{s,c} \leq 1 \;\text{ for } s = S, 0 \leq c < C \; . \tag{7}$$

3.3 Handling of Signed Multipliers

The design of signed multipliers does in general not require special consideration in the separate optimization, other than that the tiles have to be signed-capable. One important trick is to reduce sign extension vectors by negations and constants [35]. Setting the correct signedness of the individual tiles and composing the (constant) sign extension constant vector as an input to the subsequent design of the compressor tree, can be handled as a post-processing step.

In contrast, the combined optimization requires the dynamic calculation of sign extension vectors.

Therefore, the model for the combined optimization is extended with new constraints to handle the bits of the sign extension vector in the signed case. These constraints summate the individual sign extension vector constant bit contribution, represented by integer constant $h_{c,x,y}^t$, for each tile type t and particular compressor tree column c according to the position of the sub-multiplier (x, y) when its decision variable $d_{x,y}^t$ is true. As handling the complete dynamic calculation of the sign extension vector in one constraint can quickly exceed the dynamic range of the double precision float numbers used by the ILP-solver, one constraint is generated for each bit v_c of the sign extension vector V. To handle carries during the dynamic calculation of the V vector, the bits that overflow from the previous column with lower weight a_{c-1} are added in the constraint. The carries that originate from the current constraint are considered, by subtracting a_c weighted by 2 according to the higher weight of the subsequent column. There are G guard constraints ($G = 4$ showed experimentally to be sufficient) to absorb the bits v_c above the MSB $C \leq c$ of the compressor tree, which are discarded. This sum has to be equal to 0, forcing the constant bit v_c for a particular column c to be set according the column weight 2^c represented by the respective constraint within the sum of the sign extension vectors

$$\text{C8}: \quad \sum_{t=0}^{T-1} \sum_{x=0}^{w_X-1} \sum_{y=0}^{w_Y-1} h_{c,x,y}^t d_{x,y}^t + a_{c-1} - 2a_c - v_c = 0 \; . \quad \text{for } 0 \leq c < C+4$$

To account for the bits v_c in the compressor tree design, they have to be added to the C3′ constraints in the signed case, as bits that enter the first stage of the compressor tree

$$\text{C3}'' : \quad N_{s,c} = \left. \sum_{t=0}^{T-1} \sum_{x=0}^{w_X-1} \sum_{y=0}^{w_Y-1} g_{c,x,y}^t d_{x,y}^t + v_c \right\} \begin{array}{l} \text{for } s = 0, \\ 0 \leq c < C. \end{array}$$

3.4 Handling of Truncated Multipliers

Implementing the optimal design of faithfully rounded truncated multipliers as recently presented in [14] in combination with the combined optimization of tiling and compression changes and introduces several new constraints. It also requires the dynamic calculation of a combined constant bit vector to be added to the compressor tree as with signed multipliers. In faithfully rounded truncated multiplier, a constant Q is used to re-center the single sided error from the truncation and to perform the rounding as $+\frac{1}{2}$ unit in the last place (ulp) [36].

For that, constraint C1 has to be adapted to not force the placement of tiles but rather consider the truncation error dynamically, by introducing the variable $b_{x,y}$ instead of setting the constraint to one

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

The truncation error re-centering constant Q has to be constrained to be below the maximal permissible truncation error $\delta_{\widetilde{P}}$

$$\text{C9}: \qquad Q < \delta_{\widetilde{P}} \; .$$

The sum of all truncation errors and the compensation due to truncation error re-centering constant is constrained to the maximal permissible truncation error

$$\text{C10}: \qquad \sum_{x=0}^{X-1} \sum_{y=0}^{Y-1} (1 - b_{x,y}) \cdot 2^{x+y} - Q < \overline{\delta}_{\widetilde{P}} \; .$$

To realize the dynamic calculation of the error re-centering constant its individual weighted bits q_i are constrained to be equal to the constant Q

$$\text{C11:} \qquad Q = \sum_{i=l_{\text{ext}}}^{l_P-2} 2^i q_i \ .$$

To handle the new constant bits q_i in the combined optimization, they have to be added in constraints C8 to consider them in the dynamic calculation of the bits of the constant vector v_c. Beside the weighted bits of the error re-centering constant q_i, the rounding constant r bit is added, leading to

$$\text{C8}': \quad \sum_{t=0}^{T-1} \sum_{x=0}^{w_X-1} \sum_{y=0}^{w_Y-1} h^t_{c,x,y} d^t_{x,y} + a_{c-1} - 2a_c - v_c + q_i + r = 0$$
$$\text{for } 0 \leq c < C+4$$
$$\text{with} \qquad r = \begin{cases} 1 & \text{when } c = l_P - 1 \\ 0 & \text{otherwise} \end{cases}$$

In contrast to the separate optimization in [14] the average compression costs for the bits q_i do not have to be considered in the cost function, since the combined optimization considers the actual compression cost based on the utilized compressors.

3.5 Handling of Signed Truncated Multipliers

In [14] the truncated signed case was not explicitly considered and the truncation error for removing partial products was assumed to be always negative. However as the partial products at the left and bottom of the tiling board (corresponding to the MSBs of the inputs) contribute with a negative value to the final result, there is a positive tiling error when those partial products are omitted. To maintain the requirements for a faithfully rounded truncated multiplier, the negative and positive truncation errors have to be estimated and constrained to be below the error budget $\overline{\delta}_{\widetilde{P}}$ separately. While the error re-centering constant Q reduces the effective negative truncation error and thus allows for the removal of additional partial products [37], the positive error budget is reduced by applying the constant, as the positive constant is added to the positive error. The constraints for limiting the truncation error from [14] are split in two separate parts to account for the negative and positive error as follows:

$$\text{C10n:} \qquad \sum_{x=0}^{w_X-1} \sum_{y=0}^{w_Y-1} \text{sn}_{x,y}(1 - b_{x,y}) \cdot 2^{x+y} - Q < \overline{\delta}_{\widetilde{P}}$$

$$\text{C10p:} \qquad \sum_{x=0}^{w_X-1} \sum_{y=0}^{w_Y-1} (1 - \text{sn}_{x,y})(1 - b_{x,y}) \cdot 2^{x+y} + Q < \overline{\delta}_{\widetilde{P}}$$

with

$$\text{sn}_{x,y} = \begin{cases} 0 & \text{when } x \neq y \wedge (x = w_X - 1 \vee y = w_Y - 1) \\ 1 & \text{otherwise} \end{cases}$$

Here, $\text{sn}_{x,y}$ is a pre-calculated constant in the ILP-model, that defines if a partial product generates a positive or negative contribution to the final result. Hence, the constraints C10n/p are still linear despite the multiplication.

4 Results

The proposed holistic multiplier optimization scheme was implemented by extending the `IntMultiplier` operator within the open-source VHDL core generator framework (FloPoCo) [38]. The implementation is open-source and available in the FloPoCo git repository[1].

4.1 Experimental Setup

Several experiments have been performed to evaluate the scalability of the proposed combined optimization as well as the improvements compared to previous work. As a baseline, we synthesized an inferred multiplier (using the VHDL $\star$ operator of the IEEE numeric_std package). In addition, two variants of the LogiCORE IP cores of Xilinx Vivado 2020.1, one optimized for area and the other optimized for speed have been generated and synthesized. We consider the separate optimization of Fig. 1a as the state-of-the-art. For that, we used the (locally) optimal separate ILP models for tiling [4] and subsequently compression [30]. Both have been previously shown to be superior to heuristic methods in terms of resources.

A ternary adder [31], [32] is used in the reference and proposed design method as a row adder in the final stage of the compressor tree. The cost of the (1;1) compressor (wire or flip-flop) was fixed to 0.5 in the previous compression [30] while it turned out to be advantageous to set this cost to 0 as not every stage is pipelined. Gurobi v9.1.1 was used as ILP-solver for the optimizations, with a timeout of 2h for the combined and 1h for each of the separate optimization of tiling and compression. The experiments were performed on an AMD Ryzen 5 3600 CPU @ 4.00GHz (6 cores, 12 threads) and 16 GB of RAM running Linux Mint 19.2 LTS. Xilinx Vivado 2020.1 was used for the synthesis experiments for a Kintex-7 target FPGA (xc7k70tfbv484-3). All synthesis runs include place&route and were performed with the option `flatten_hierarchy=none`, because when Vivado performed a full or partial rearrangement of the circuit that relies on a specific LUT-mapping, it resulted in a considerably higher LUT-cost. The timing data for the critical path delay (CPD) was measured between additional double wrapper registers at the input and output signals of the circuit to achieve realistic conditions, like when the multiplier is integrated in a larger design.

Several experiments were performed covering all combinations of unsigned/signed, truncated/non-truncated multipliers using LUT-only, $2 \times k$ and DSP multiplier tiles, as represented on the left of Table 4. The input word sizes were set to identical values ($w_X = w_Y$) and evaluated between 2 and 32 bits. The word size of the result w_R was either left at full precision or set to $w_R = w_X = w_Y$ in the truncated case.

4.2 Scalability of the Method

As the proposed combined optimization model is considerably more complex, the first question to answer is up to which problem sizes it is realistically able to provide solutions. There are two limiting factors for the maximal model size the method is able to process. First, the rise

1. See link to the git at main page of `http://www.flopoco.org`

in complexity with model size results in an increase in the solver runtime. Second, in the larger cases numeric problems can occur due to the limited dynamic range of the floating point format used by the ILP solver.

Most solvers like Gurobi use double precision to represent real numbers. Due to the accumulation of rounding errors, a weighted sum of variables close to zero, representing a binary false, may evaluate to >1 (*true*) after rounding. This and similar issues are well known but unavoidable; the recommendation is to limit the dynamic range of variable or constants to 10^4, which is not always possible [39].

So it was first evaluated, up to which size the solver is capable to find an optimal solution within the given timeout. The right side of Table 4 shows the multiplier size with the first occurrence of this timeout in the combined and separate approach and the first occurrence of the numeric problems for several combinations of signedness, truncation and different multiplier tiles. The combinations are denoted with a '1' if applicable and otherwise with '0', while the optimization to minimize stages (as 1st objective) is marked with 'S' and to minimize LUTs with 'L'. While timeouts lead to feasible solutions that are not proven to be optimal, a numeric issue is typically catastrophic (the solution can not be used).

As expected, the timeouts in the combined model occur earlier compared to the separate optimization, but the differences are not large. The solving times show initially a slow rise with the multiplier size and the resulting partial products until they abruptly surge beyond the timeout limit. For example, in the case described by the first row of Table 4, the small word sizes of $w_X = 2 \dots 10$ where solved in less than a minute while the timeout of 2 hours was not sufficient for $w_X > 14$. This often coincides with the multiplier size, where an additional stage is required to compress the partial products.

It can be seen that the use of large multiplier tiles like the $2 \times k$ or the DSP reduces solving times (due to the lower required number of stages) and thus extends the multiplier size where the timeout is reached. As expected, the truncated case reduces the complexity of the model as fewer partial products have to be computed, while the dynamic calculation of the constants vector on the compressor tree for the sign extension and the rounding and error re-centering constant in the truncated case, cause some overhead in terms of complexity, when comparing the solving times between the signed and unsigned case.

Some numerical problems with the ILP solver were encountered in the signed and truncated case during the experiments. In the truncated case the truncation error is dynamically considered by calculating the weighted sum of the omitted partial products and constrained to be below the error budget. This creates a large dynamic range between the MSBs and the LSBs of the truncation error, causing rounding errors and warnings by the solver, beyond a certain multiplier size.

In summary, the experiments show, that our holistic approach scales well to multiplier problem sizes that are of practical use. Note that even if a timeout is encountered, and the result is feasible but not guaranteed to be optimal, its objective is often still better as shown in the following.

TABLE 4: First occurrence of optimal timeout and numerical problems of the solver

case			tiles		multiplier size w_X=w_Y		
		minimize	use	use	combined		separate
signed	truncated	stages/LUT	DSP	2×k	timeout	num. prob.	timeout
0	0	S	0	0	14	–	16
0	0	S	0	1	15	–	21
0	0	S	1	0	26	–	25
0	0	S	1	1	26	–	29
0	0	L	0	0	14	–	16
0	0	L	0	1	15	–	21
0	0	L	1	0	25	–	25
0	0	L	1	1	26	–	29
0	1	S	0	0	16	–	–
0	1	S	0	1	15	–	–
0	1	S	1	0	28	–	–
0	1	S	1	1	28	–	–
0	1	L	0	0	16	22	17
0	1	L	0	1	15	27	22
0	1	L	1	0	28	–	–
0	1	L	1	1	28	–	–
1	0	S	0	0	13	25	17
1	0	S	0	1	15	–	17
1	0	S	1	0	21	28	24
1	0	S	1	1	25	20	28
1	0	L	0	0	14	23	17
1	0	L	0	1	15	–	17
1	0	L	1	0	22	24	26
1	0	L	1	1	25	20	28
1	1	S	0	0	14	27	18
1	1	S	0	1	15	28	25
1	1	S	1	0	27	29	–
1	1	S	1	1	28	28	–
1	1	L	0	0	14	26	18
1	1	L	0	1	15	–	25
1	1	L	1	0	28	–	–
1	1	L	1	1	28	–	–

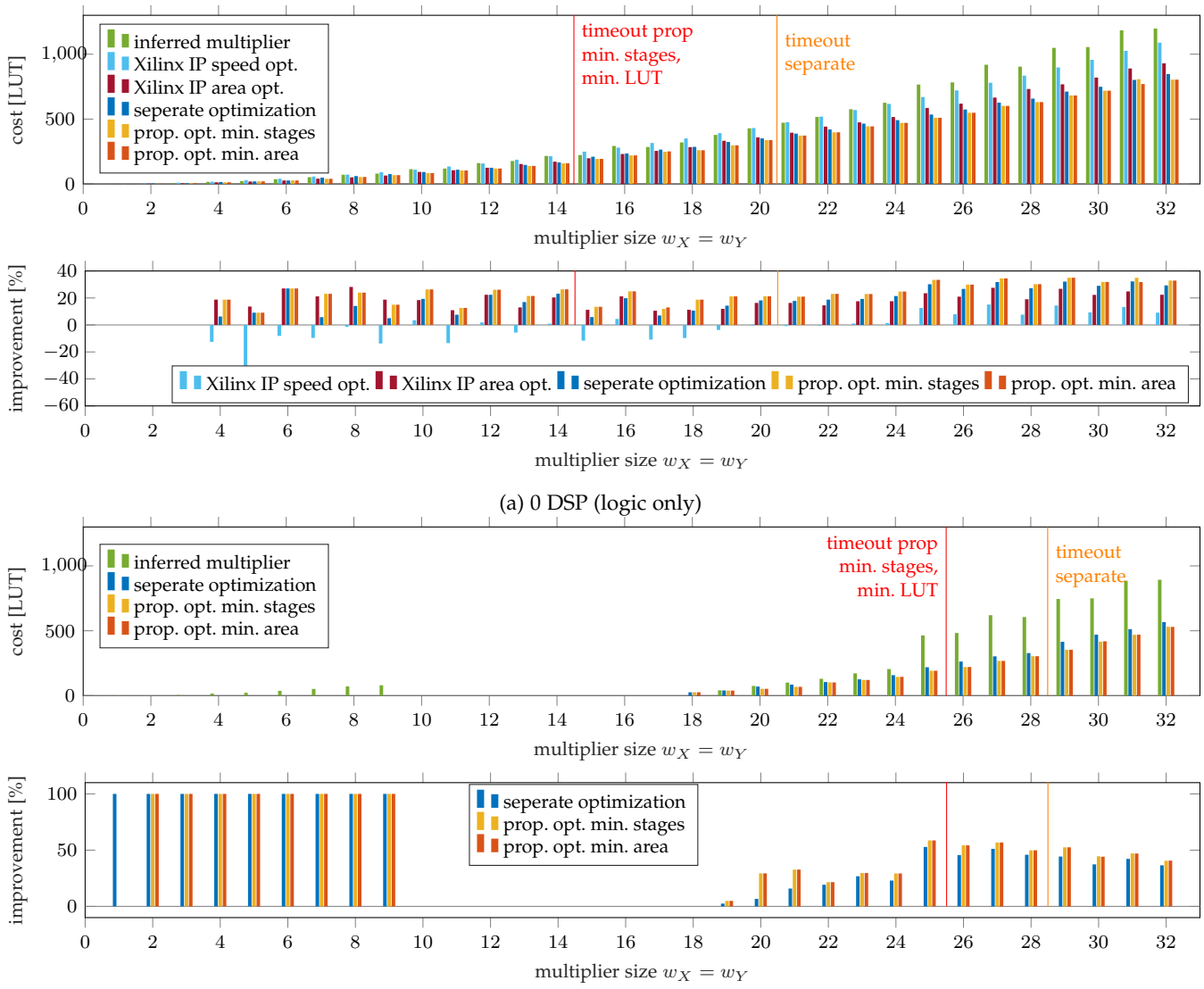
4.3 Quality of Results, LUT resources

Synthesis experiments have been performed for the combinations listed in Table 4 (including place and route) up to word sizes of 32 bits. To cover the most relevant cases we show the synthesis results of the unsigned multipliers with full precision used in integer applications and the signed truncated cases used in digital signal processing or machine learning applications. For both cases, logic only or one DSP was allowed with the 1st objective to either minimize the stages or the LUTs. It has to be noted that the other combinations of Table 4 show qualitatively similar trends.

4.3.1 LUT improvements

Figs. 5 and 6 show the utilized LUT resources and the relative improvement compared to the inferred multiplier for the logic only solution (figs. 5a, 6a) as well as using one DSP block (figs. 5b, 6b). Using more DSPs would result in circuits that require only a few LUTs and hence would not provide an effective case to evaluate the improvements over the separate optimization. The Xilinx IP-cores are not included in Fig. 5b and 6b, as they do not allow to use a particular number of DSPs. To obtain the LUT-values, the reference methods were synthesized with 1 pipeline stage, while the designs from FloPoCo have up to 3 stages, as needed for the target frequency of 400 MHz.

It can be observed as a general trend, that the inferred multiplier requires the most resources, followed by the variants of the Xilinx IP-cores optimized for speed and then for area. The methods based on tiling and compressor tree



(a) 0 DSP (logic only)

(b) 1 DSP

Fig. 5: Synthesis results for multiplier sizes $w_X = w_Y = 2..32$ (unsigned) for different objectives and DSPs

usually outperform those with respect to the LUT resources while the proposed holistic combined optimization almost always delivers the best results. The average improvements with respect to the inferred multiplier in Fig. 5a are -1.1% for the IP-core optimized for speed and 18.9% for area, 18.9 % for the previous separate optimization and 23.9% for the proposed method optimized for stages and 24.0% for area, respectively. Using one DSP (Fig. 5b) and ignoring the small cases with $w < 17$ bit that only use fractions of the DSP, the average improvements are 32.1% for the previous separate optimization and 39.4% for the proposed methods.

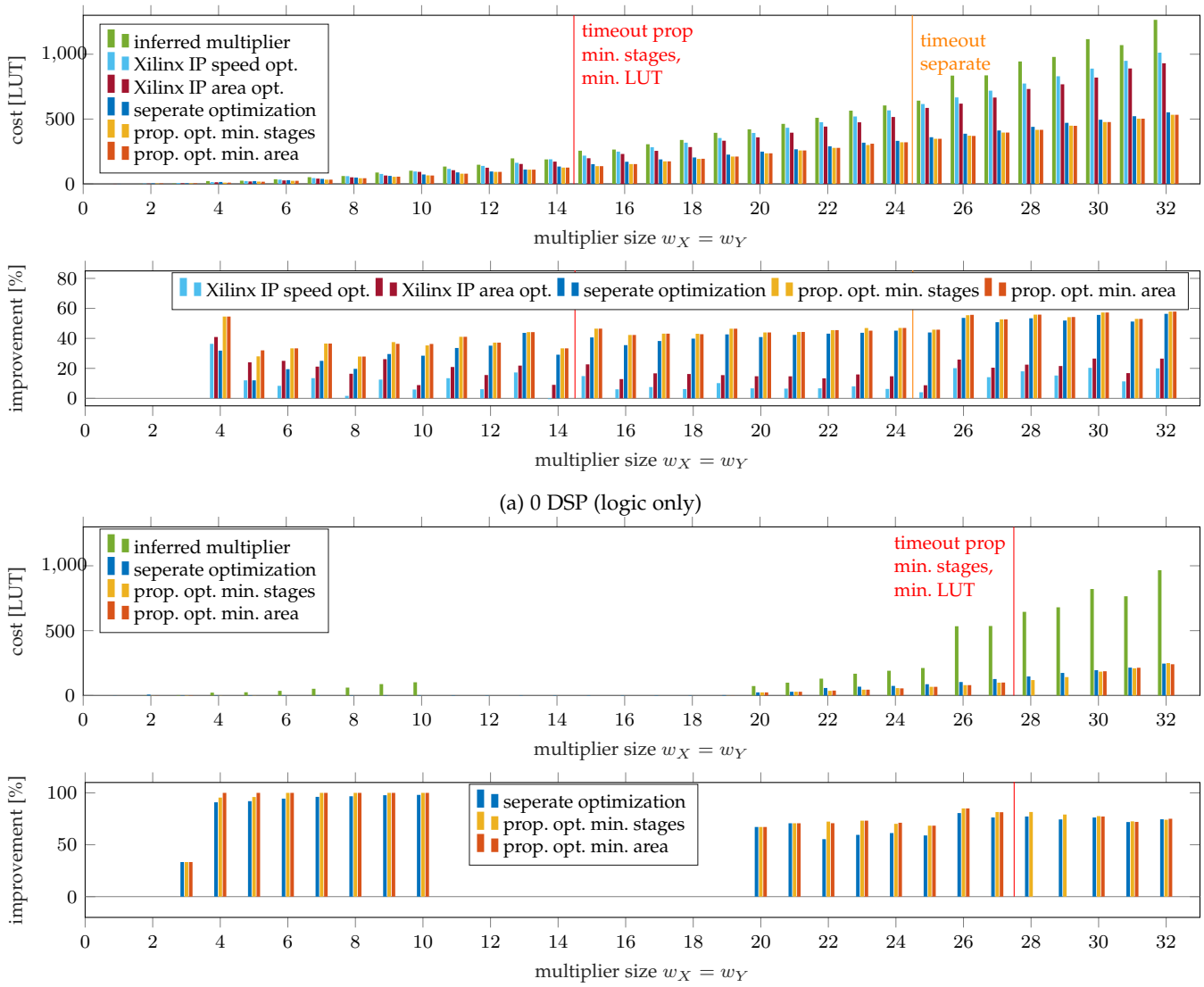
It has to be noted that the comparison with the inferred multiplier and the IP-cores in the truncated case of Fig. 6 is not fully fair as they do not support faithful rounding and had to be truncated leading to considerably higher realization costs. The average improvements in this case (Fig. 6a) are (in the same order as above) 11.3%, 19.2%, 39.2%, 44.5% and 44.6%, respectively. Using one DSP (6b) the average improvement is 64.6% for the previous separate optimization and 69.5% for the proposed method optimized for stages and 73.8% for area.

It can be concluded that the cost model within the combined optimization is a realistic representation of the actual LUT costs. Another observation is that there are still many cases where a better solution was found compared to the separate optimization even when no optimal solution was found (right of the red "timeout" line).

4.3.2 LUT vs. stage minimization

To further investigate the differences between the proposed variants for minimizing area and stages, Fig. 7 shows the improvements of the proposed method minimizing area and stages relative to the previous tiling method as well as their stage count for the unsigned case without DSP and 2×k.

The results in terms of the number of LUTs are relatively similar when minimizing the stages compared to minimizing the LUTs, however, in several cases one stage can be saved ($w_X = 13, 27 \ldots 30$ in Fig. 7b), resulting in a shorter (pipeline) latency or delay. An example of this is shown in Fig. 8. Although the tiling is slightly more costly in the



(a) 0 DSP (logic only)

(b) 1 DSP

Fig. 6: Synthesis results for multiplier sizes $w_X = w_Y = w_{\text{Out}} = 2..32$ (signed, truncated) for different objectives and DSPs

combined approach, it leads to a flatter dot diagram that can be compressed with a single stage and lower overall costs. In some cases ($w_X = 13, 16, 27, 29, 30$ in Fig. 7) a solution with less LUTs can be found using one additional stage.

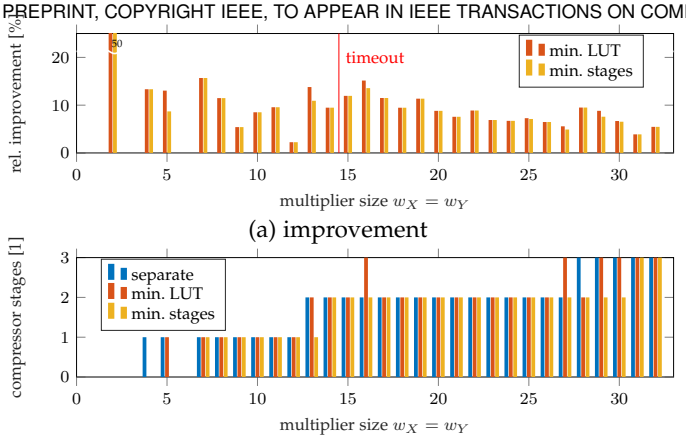
The LUT improvements in Fig. 7 of the combined approach in which an optimal solution was found (left of the red "timeout" line) are in the range of 0...15% with one exception: In the very small 2×2 multipler, 50% improvements are achieved (2 compared to 4 LUTs) by using a single 2×3 tile that does not require any compression. However, in the separate optimization two 1×2 multipliers are cheaper for the tiling (see Table 1) but require additional cost for compression.

4.4 Quality of Results, Delay

Thanks to the automatic pipelining in FloPoCo, the delay results are very similar for all experiments. Exemplarily, Fig. 9 shows the critical path delay for the unsigned multiplier case. The reference methods have been pipelined with 3 stages in this experiment and the proposed methods uses 0...2 pipeline stages when minimizing the stages and 0...3 stages when minimizing the LUTs. Apart from the inferred multiplier, only a slight increase in the critical path delay can be noted proportional to the multiplier size, from 3.5 ns to 5 ns, because of increasing routing delays for larger circuits that are only approximated in the automatic pipelining in FloPoCo.

4.5 Packing Density

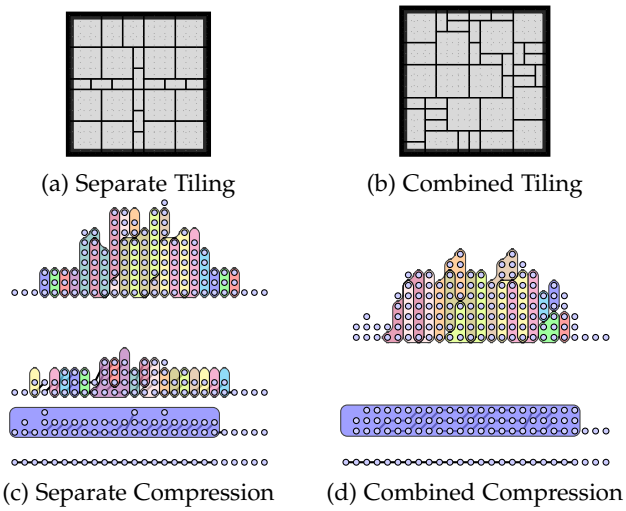
Apart from the utilization of LUT resources, it is much harder to measure the used routing resources as there are no tool reports available for that. A simple but elegant way is to evaluate the *packing efficiency*, by filling the FPGA with several instances of the same design until it can not be routed anymore [10]–[12]. Packing experiments for different multiplier sizes (8×8, 16×16, 32×32) were performed for unsigned full precision multipliers of the proposed method (for minimizing area) and the reference architectures (inferred VHDL multiplier as well as Xilinx IP core optimized



(a) improvement

(b) compressor stages

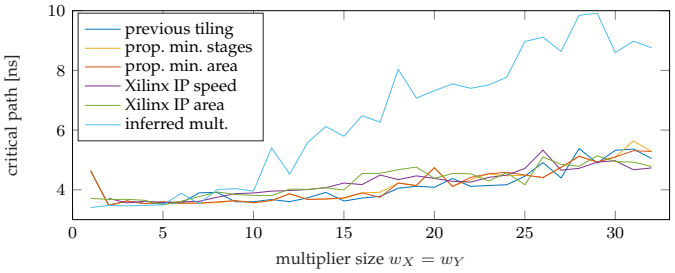
Fig. 7: Improvement to previous separate optimization after synthesis and stages count for different objectives



(a) Separate Tiling (b) Combined Tiling

(c) Separate Compression (d) Combined Compression

Fig. 8: Example of an unsigned 13×13-multiplier generated with separate and combined optimization



(a) critical path delay

Fig. 9: Critical path delay for unsigned multiplier size $w_X = w_Y = 2..32$, with 0 DSP for various methods

for speed and area). All multipliers utilize one pipeline stage to have similar conditions. They are connected in series, where the $2W$-output bits are connected to the two inputs of W bits each to mimic a realistic interconnect. The `flatten_hierarchy=none` configuration makes sure that no optimization happens between the different instances.

There is no documentation about the multiplier structure that is used by the inferred or IP-based multipliers of Xilinx.

TABLE 5: Results from packing experiment

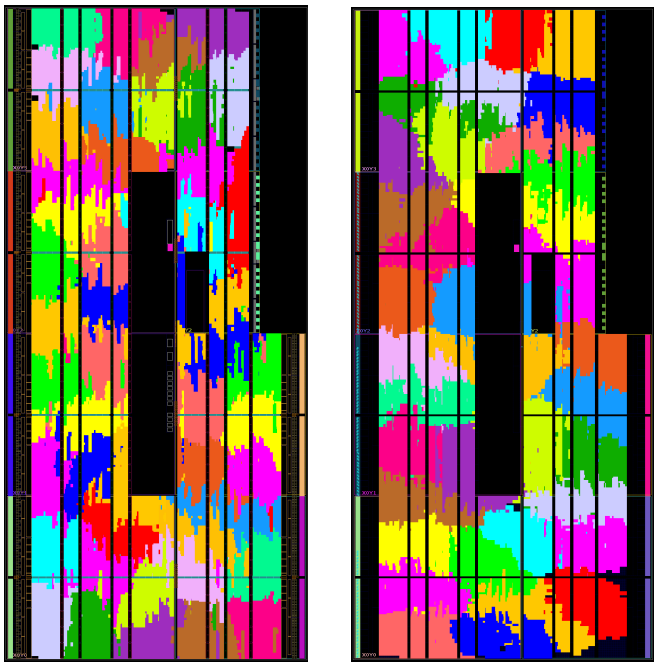
size	design	single mult.		#mult/FPGA		Utilization [%]	
		#LUTs	CPD [ns]	theory	actual	Slice	LUT
8×8	proposed	54	4.7	759	732	100.0	94.6
8×8	IP speed opt.	72	4.3	569	479	98.1	84.1
8×8	IP area opt.	51	6.7	803	640	100.0	79.6
8×8	inferred mult.	71	8.1	577	471	100.0	68.9
16×16	proposed	225	7.7	182	173	99.8	92.9
16×16	IP speed opt.	280	5.9	146	129	96.9	88.1
16×16	IP area opt.	231	8.3	177	143	99.5	80.6
16×16	inferred mult.	293	12.0	139	138	99.5	82.8
32×32	proposed	800	9.7	51	46	99.4	89.8
32×32	IP speed opt.	1089	8.8	37	34	95.5	90.3
32×32	IP area opt.	930	9.3	44	37	98.3	83.9
32×32	inferred mult.	1198	16.4	34	27	99.0	78.9

However, from the schematics obtained after synthesis & implementation, one can derive details of their structure. The inferred multiplier consists of multiple stages of logic and carry chains. This might explain why this is among the slowest multipliers. The Xilinx IP cores use a single stage of logic to compute partial products of up to six inputs, followed by pipeline registers and an adder tree. The adder tree is built from regular two-input adders in the speed opt. variant while ternary adders are used in the area opt. variant. This is plausible as two-input adders are known to be faster but also less area efficient than ternary adders (see Table 3).

Table 5 shows the results of the packing experiment. The used LUT resources and the CPD of the individual single multipliers are summarized in the third and fourth column. From these individual LUT resources, the number of multipliers that can be mapped theoretically (column "theory") is obtained by dividing the number of available LUTs (41,000 for the selected FPGA) by the required LUT resources. The actually achieved maximum number of multipliers per FPGA is shown in column "actual". The last two columns show the percentage Slice and LUT utilization. As expected, each method achieves a slices utilization close to 100% but a lower LUT utilization down to about 70%. It can be observed that the proposed method achieves a significant better LUT utilization compared to previous designs. In every case, the most instances can be fitted with the proposed method.

We only speculate why our proposed method actually has a higher packing density compared to the reference architectures. One reason might be that, in contrast to the reference methods, many of our computational units (most GPCs and the $2 \times k$ multiplier) are designed to fully utilize a slice as a single unit. Hence, no additional routing between LUTs and the fast carry chain is necessary. This also makes sure that each LUT in a slice is actually used.

Some more details can be obtained from inspecting the floorplans of the target device for the different designs. Fig. 10 shows the floorplan packed with the maximal possible instances of a 32×32-multiplier generated by the proposed design in Fig. 10a (46 instances) and the Xilinx IP core optimized for area in Fig. 10b (37 instances). The slices of individual multipliers are highlighted with different colors



(a) proposed (b) area opt. IP

Fig. 10: Floorplan of the FPGA filled with the maximal possible instances of the respective 32×32-multiplier design

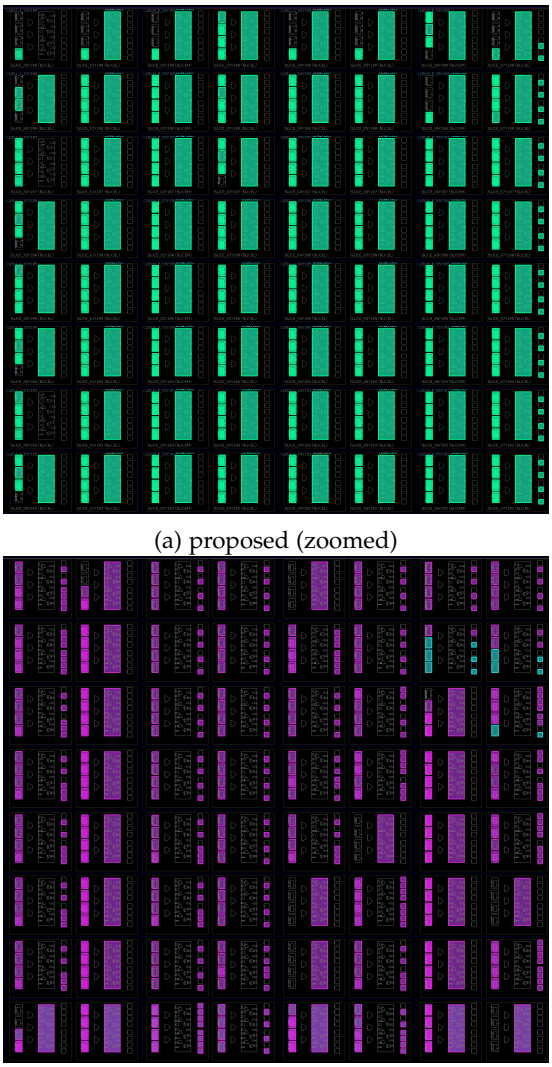
(the colors are repeating). It can be seen, that components of the instances are placed predominately locally clustered. Components that are immediately connected are also placed close to each other. The instances of the proposed design (Fig. 10a) are noticeably smaller on the chip, which could contribute to their outline not being as defined as in the area optimized IP (Fig. 10b). Other than at the edges where placing components would result in long signal paths there are few unused regions in both cases.

Fig. 11 shows a magnified view of the top left corner of the floorplan for the proposed and IP-based designs, respectively. This shows detailed LUTs, carry-chain and flip-flops resources from left to right (repeating). Similar utilization patterns appear consistently over the whole floorplan. As expected, the proposed design in Fig. 11a uses the available fast carry-chains considerably more extensively. This is due to the heavy use of the efficient $2 \times k$ multipliers as well as GPCs in the compressor tree.

In contrast, the area optimized IP in Fig. 11b uses the fast carry-chain less often and when it is used the corresponding LUTs in the slice sometimes remain un-utilized. Apart from that, the position of the pipeline stage in the area optimized IP requires many more flip-flops. While most flip-flops come for free as they are located in the same slice, in some cases it seems to hinder the use of the carry-chains within the same slice. This might explain the lower packing density.

5 Conclusion

The proposal demonstrates how the previously separately considered tiling, compressor tree and final adder design can be transformed to a combined, holistic optimization approach. Extensions for the design of signed and/or truncated multipliers are provided, covering most practical use



(a) proposed (zoomed)

(b) area opt. IP (zoomed)

Fig. 11: Zoom of the floorplan of Fig. 10

cases in which multiplications occur. It was experimentally shown that a globally optimal multiplier design for FPGAs is feasible and scales up to relevant multiplier sizes. Any improvements in future ILP solvers regarding solving times or numerically robustness will also translate to larger multiplier sizes. Even beyond solver timeouts, typically significant resource reductions could be achieved.

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, 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.
[12] 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.
[13] 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.
[14] A. Boettcher, M. Kumm, and F. de Dinechin, "Resource optimal truncated multipliers for FPGAs," in *IEEE Symposium on Computer Arithmetic (ARITH)*, 2021.
[15] 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.
[16] 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.
[17] 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.
[18] 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.
[19] 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
[20] M. Kumm and P. Zipf, "Efficient high speed compression trees on Xilinx FPGAs," in *MBMV*, 2014.
[21] 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.
[22] 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.
[23] 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.
[24] 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.
[25] 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.
[26] L. Dadda, "Some schemes for parallel multipliers," in *Alta Frequenza*, vol. vol. 34, 1965, pp. pp. 349–356.
[27] K. A. C. Bickerstaff, E. J. Swartzlander, and M. Schulte, "Reduced Area Multipliers," ser. Application-Specific Array Processors, 1993. Proceedings., International Conference on. Application-Specific Array Processors, 1993 Proceedings, International Conference on, 00 1993, pp. 478 – 489.
[28] 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.
[29] 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.
[30] M. Kumm and J. Kappauf, "Advanced compressor tree synthesis for FPGAs," *IEEE Transactions on Computers*, vol. 67, no. 8, pp. 1078–1091, 2018.
[31] 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.
[32] G. Baeckler, M. Langhammer, J. Schleicher, and R. Yuan, "Logic Cell Supporting Addition of Three Binary Words," *US Patent No 7565388, Altera Coop.*, 2009.
[33] W. Xiao, W. Qian, and W. Liu, "Gomil: Global optimization of multiplier by integer linear programming," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 374–379.
[34] 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.
[35] M. J. Flynn and S. F. Oberman, *Advanced Computer Arithmetic*, ser. John Wiley & Sons Inc. John Wiley & Sons Inc., 2001.
[36] 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.
[37] 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.
[38] 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.
[39] L. Daniel Espinoza, Gurobi Optimization, "Gurobi guidelines for numerical issues," 2018. [Online]. Available: https://www.gurobi.com/resource/numerical-issues-webinar/

Andreas Böttcher received the B. Eng. degree in electrical engineering from the University of Applied Sciences Fulda, Germany, in 2014 and a M. Eng. degree in 2017. From 2014 to 2019 he worked in multiple research projects at Fulda university focused on control design and modeling of nonlinear systems and digital modulation. Since 2019 he is working in a research project with the main objective to optimize multiplication on reprogrammable logic.

Martin Kumm received the Dipl.-Ing. degree in electrical engineering from the University of Applied Sciences Fulda, Germany, and the Technical University of Darmstadt, Germany, in 2003 and 2007, respectively. From 2003 to 2009, he was with GSI Darmstadt, working on digital RF control systems for particle accelerators. In 2015 he received his Ph.D. (Dr.-Ing.) degree from the University of Kassel, Germany. He is currently a Professor for Embedded Systems at the Fulda University of Applied Sciences, Germany. His research interests are arithmetic circuits and their optimization as well as high-level synthesis, all in the context of reconfigurable systems.