Evaluation of Different Manual Placement Strategies to Ensure Uniformity of the V-FPGA*

[redacted][0000−0000−0000−0000], [redacted][0000−0000−0000−0000], and [redacted][0000−0000−0000−0000]

1 [redacted]

2 [redacted]

Abstract. Virtual FPGA (V-FPGA) architectures are useful as both early prototyping testbeds for custom FPGA architectures, as well as to enable advanced features which may not be available on a given host FPGA. V-FPGAs use standard FPGA synthesis and placement tools, and as a result the maximum application frequency is largely determined by the synthesis of the V-FPGA onto the host FPGA. Minimal net delays in the virtual layer are crucial for applications, but due to increased routing congestion, these delays are often significantly worse for larger than for smaller designs.
To counter this effect, we investigate three different placement strategies with varying amounts of manual intervention. Taking the regularity of the V-FPGA architecture into account, a regular placement of tiles can lead to an 37 % improvement in the achievable clock frequency. In addition, uniformity of the measured net delays is increased by 39 %, which makes implementation of user applications more reproducible. As a trade-off, these manual placement strategies increase area usage of the virtual layer up to 16 %.

Keywords: FPGA · EDA · Placement · Virtual FPGA

1 Introduction

In recent years, virtual Field Programmable Gate Arrays (FPGA) architectures (V-FPGAs) have been been introduced in academia [2]. Unlike common commercial and academic FPGAs, the V-FPGA is an FPGA architecture layered onto a base FPGA architecture: A commercial host FPGA architecture is synthesized for a silicon chip target, and the V-FPGA layer is synthesized for that host FPGA architecture. The virtual layer is implemented as a bitstream to be programmed onto the host FPGA. User applications are synthesized using a custom toolchain for the V-FPGA layer and the resulting application bitstream is programmed onto it. V-FPGA architectures have been applied for three main

* [redacted]

use cases: First, as an abstraction layer, providing a common bitstream format independent of commercial FPGA architectures. This allows using features such as partial dynamic reconfiguration on FPGAs which do not natively support this [6]. Secondly, V-FPGAs can be used for FPGA architecture research: Novel ideas can be integrated in the architecture and tested on a hardware implementation, which can provide additional insight compared to simulation. This becomes especially useful when investigating heterogeneous System-on-Chips (SoCs) solutions, which may combine processing systems and reconfigurable logic [2,4]. Thirdly, using the V-FPGA as a basic FPGA architecture: Here it is realized using standard cell synthesis for silicon targets and can be used to evaluate the usage of different transistor technologies in FPGAs.

Targeting FPGAs instead of silicon, synthesizing and implementing a V-FPGA requires different considerations than normal FPGA architectures. Most notably, whereas FPGAs are usually designed in a custom or semi-custom way based on tiles, V-FPGA architectures are written in standard hardware description language. Usual FPGA synthesis is used, trying to minimize critical path length to meet a target clock frequency. The V-FPGA code introduces a hierarchy of components and supports the concept of tiles. These tiles are a logical representation and used for silicon targets, but are not considered in the synthesis, mapping and placement onto the host FPGA. This leads to various issues:

1. The final target clock frequency must be given without knowing the user application. Usually an “as fast as possible” solution is used, but for common timing driven synthesis and placement, this means manually increasing the frequency until the design fails to synthesize, which is a time-intensive process.
2. The V-FPGA architecture is not uniform: For example, the delay of a connection between two neighboring tiles depends on their location on the FPGA. Whereas the maximum delay can be used to describe the architecture for the synthesis toolchain targeting the virtual layer, this limits the ability to overclock user applications: As routing is not deterministic, a user application may be synthesized to different tiles in different synthesis runs and the real timing slack varies. To achieve deterministic results, the delays of the V-FPGA layer need to be uniform.
3. As any FPGA architecture consists of configurable elements and multiplexers, it is possible to form combinational loops on these architectures. Such loops break timing analysis and need to be marked accordingly using disable timing constraints, otherwise timing analysis obtained from commercial tools is wrong.
4. As the FPGA synthesis tools are not aware of the structure of the V-FPGA architecture, they are essentially just placing a large network. This may increase runtime and we postulate it further leads to worse critical path delay for large designs, as the effort to place the whole design becomes too large to yield results similar to a grid-based layout.

To address these problems, we investigate manual placement strategies forcing the mapped design onto certain locations on the host FPGA. First, we describe metrics measuring the variation in delays of similar V-FPGA nets (uniformity), delay and area. Secondly, we explain three strategies to constrain the tiles in the V-FPGA in a grid-like layout, how the size and form of the tiles has to be determined and how we place cells within the tiles. In the end, we evaluate the results based on the introduced metrics. Results show that the three different strategies increase uniformity and lead to decreased maximum delay, at the cost of slightly increased area.

2 Related Work

For previous research in placement algorithms for FPGA, we can differentiate two research areas: Placement algorithms for generic FPGA applications and manually guided placement for specific applications. Research on generic placement algorithms has largely been conducted using academic open source tools. The most-widely used of these tools is Verilog to Routing (VTR), which consists of three main tools: Odin II for synthesizing circuits designed in Verilog into generic Lookup Table (LUT) resources, ABC for technology-mapping those resources into architecture specific LUTs, and Versatile Place and Route (VPR) for packing, placement, and routing. Research led to the introduction of new techniques and algorithms for placement and packing, focusing on different cost targets such as timing driven, routing-driven or runtime-driven: [7] introduces an algorithm named adaptive range-based Simulated Annealing (ARBSA). It provides an adaptive approach to choose the neighborhood for each block according to the nets it belongs to. The result shows a 1.78X runtime speed up, 10 % reduction on wire length and 2 % reduction on the critical path with path timing-driven optimization. Research in this area proposes general algorithms, trying to find good solutions regardless of the target application. Results are therefore usually shown relative to the VTR algorithms.

The second group of research uses the concept of manual placement, ranging from guidance to completely manually placed designs. For example, [1] introduced a sea-of-gates architecture called Triptych. It aims to reduce the significant cost paid for routing in standard FPGAs, replacing the logic blocks with Routing and Logic Blocks (RLBs). RLBs perform both logic and routing tasks, allowing a tradeoff between logic and routing resources on a per mapping basis. Using manual placement made this architecture yield a logic density improvement of up to a factor of 3.5 over commercial FPGA's automatic placement. As another example, Shi et al. analyzed manual placement for their specific FPGA application [5]. It shows that using manual placement leads to a compact and optimized design with shorter nets, reducing propagation delay up to 25 %.

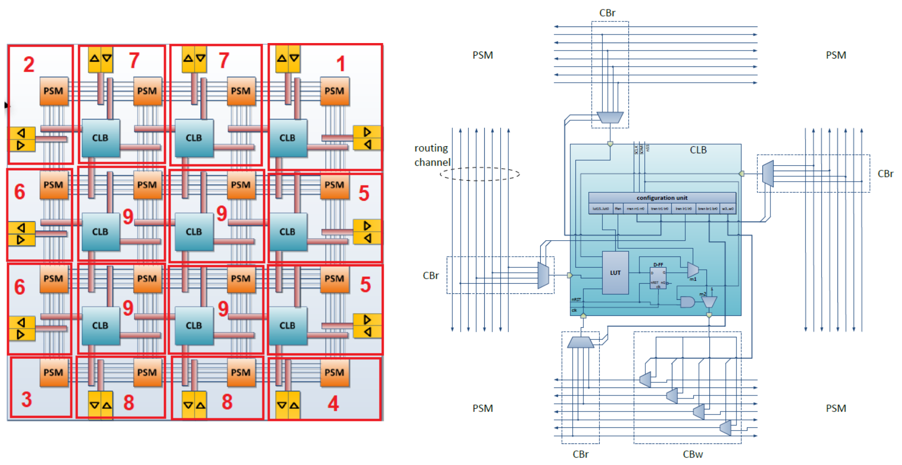
Whereas generic algorithms can be used to implement the V-FPGA, they have to be reimplemented in the Vivado synthesis flow, as they originally target VPR. In addition, none of these algorithms takes the regular structure of the V-FPGA into account. The presented application specific manual placement methods on the other hands side, do not directly translate to the V-FPGA structure. They need to be heavily modified to be usable for this use case. In order to improve net delay in the V-FPGA, we therefore investigate specific custom placement methods based on similar ideas, but explicitly considering the V-FPGA architecture and regularity.

3 Background

The placement strategies to be introduced make use of the regular structure of the V-FPGA. As such, they are dependent on both the V-FPGA architecture, as well as on the host FPGA architecture. The strategies have been designed to work with a certain parameter variability in these architectures, but certain assumptions have to be made.

3.1 V-FPGA Architecture

Figure 1a shows the V-FPGA architecture and the arrangement into certain types of tiles. In its simplest configuration, the V-FPGA consists of tiles of Con-



(a) Tile Distribution. (b) CLB Implementation.

Fig. 1: V-FPGA architecture details: a): Tile Distribution and top-level architecture. b): CLB implementation and routing channels.

figurable Logic Blocks (CLBs), Programmable Switch Matrices (PSMs) and I/O Blocks (IOBs), which are drawn as triangles on the peripheral area. A single tile can contain all of these elements (type 5 and 7), all but the IOB (type 9, the central tiles), PSM and IOB (type 2, 4, 6, 8), PSM and two IOBs (type 1), or only the PSM (type 3). Because of their differences in orientation or structure — and therefore layout —, individual tile types will have to be handled differently. Figure 1a also shows relevant delays for the final architecture: Apart from intra-block delays such as delays in the CLB, these consist of nets in the global interconnect channels. Channel nets can be divided into multiple segments, if they're divided by connection boxes attached to IOB or CLB units (figure 1b). Extraction of delays for VPR focuses on these atomic segments, but for assessment of placement results in this publication, we will always consider complete PSM-to-PSM and CLB-to-PSM paths.

3.2 Xilinx 7-series Architecture

In this work, we primarily target Xilinx 7 Series devices, although the methods are generic and adapt to other host FPGA architectures. For manual placement, the most important aspect of the host FPGA architecture is its uniformity: Placing the V-FPGA in a uniform way is a simple task if the host FPGA is uniform, but this is not the case for recent commercial FPGAs. As an example, consider trying to place the V-FPGA across the whole area of a 7-series device: Delays between two tiles on two sides of a hard IP column will be significantly larger than delays between two tiles not separated by hard IP. Manual placement strategies have to take this effect into account.

3.3 Metrics

To evaluate and assess the results, we first introduce comparison metrics. The commonly used metrics — delays, uniformity and area — have a specific meaning for V-FPGA targets and are crucial for V-FPGA applications.

Uniformity is a measurement of local delay variation across the V-FPGA structure. Placing every tile in the same way, the V-FPGA is a uniform structure, which in theory could be placed uniformly on the host FPGA. In practice, not all tiles have exactly the same internal structure and non-uniformity of the host FPGA architecture will further degrade results. To address this, our definition of uniformity divides the V-FPGA into N_C sets, where each set represents one column. We also group nets into classes, so that the same nets in different tiles are in a class. Uniformity then targets rows within a set, but no uniformity is guaranteed between these sets. This definition is formalized in the following equations:

$$\mu_{c,n} = \frac{1}{N_R} \sum_{r=1}^{N_R} t_{c,r,n} \tag{1}$$

$$\sigma_{c,n}^2 = \frac{1}{N_R} \sum_{r=1}^{N_R} \left(t_{c,r,n} - \mu_{c,n}\right)^2 \tag{2}$$

$$\overline{\sigma} = \frac{1}{N_C N_N} \sum_{c=1}^{N_C} \sum_{n=1}^{N_N} \sqrt{\sigma_{c,n}^2} \tag{3}$$

$$\overline{c_v} = \frac{1}{N_C N_N} \sum_{c=1}^{N_C} \sum_{n=1}^{N_N} \frac{\sqrt{\sigma_{c,n}^2}}{\mu_{c,n}} \tag{4}$$

Equation (1) provides the arithmetic mean $\mu_{c,n}$ of the delays (t) of a net class (n) in a column (c), calculated over the V-FPGA rows. $\sigma_{c,n}^2$ then calculates the variance for a net class in a certain column over the rows. This is further used in $\overline{\sigma}$ to calculate the arithmetic mean of the standard deviations of all net classes in all columns. $\overline{c_v}$ provides the arithmetic mean over the coefficient of variation of all net classes in all columns. Whereas the standard deviation is an absolute value and therefore depends on the mean of the delays, the coefficient of variation provides a relative measurement. As the delays in the host FPGA are largely discrete (e.g. fixed delays in LUTs), it is expected that relative delays can not be reduced further at some point. Because of this, we use $\overline{\sigma}$ to guide the design of our strategies and for evaluation of practically achievable uniformity. $\overline{c_v}$ is used to judge the quality of results for V-FPGA: As a smaller delay τ allows to put more logic elements in a path at the same frequency for V-FPGA applications, a constant standard deviation leads to reduced certainty of the number of V-FPGA logic elements in the path. A constant relative value $\overline{c_v}$ signifies unchanged conditions for the V-FPGA application synthesis.

Delay in the V-FPGA is a measurement that determines the final achievable application frequency. Equations to find the maximum delay among the building blocks are given below:

$$\tau_{c,n} = \max_{r \in \{1..N_R\}} t_{c,r,n} \tag{5}$$

$$\tau = \max_{c \in \{1..N_C\}} \max_{n \in \{1..N_N\}} \tau_{c,n} \tag{6}$$

$\tau_{c,n}$ selects the worst delay of a net class (n) in a column (c), calculated over the V-FPGA rows. τ uses this to find the absolute maximum delay over all columns and all net classes in the design, providing a single value for evaluation.

Area is measured in number of host FPGA CLBs used by the V-FPGA design. Used CLBs in the design do not solely consist of the V-FPGA building blocks: It also includes CLBs that are constrained to be explicitly not used in placement, optimizing the placement regularity. For partition blocks, their size may need to be slightly more than the minimum required area, as aiming for utilization ratio of 100 % may cause routing to fail. 87 % utilization rate is the default target chosen by Vivado and is our starting point for manual placement.

3.4 Methodology

Our overall approach to implement and evaluate the custom placement strategies for the V-FPGA consists of three steps: At first, the V-FPGA code is synthesized in Xilinx Vivado. Secondly, we run custom Tool Command Language (TCL) scripts on the synthesized design, adding various timing and location constraints. The third step is needed to evaluate the results and consists of running a custom TCL script to extract timing and area information.

Synthesis Synthesis largely follows the Vivado default strategy. To ensure proper conditions for the TCL script, some settings are adjusted: The *flatten_hierarchy* option is changed from the default *rebuilt* to *none*. As V-FPGA designs often provide customizable parameters [3], the default option *rebuilt* leads to unpredictable signal names when these parameters change. Changing this setting can also affect optimization across hierarchy levels. To limit the impact of this, the implemented design was analyzed manually and some optimization have been carried out manually in the VHDL source code.

Constraining the Design As Vivado analyzes every possible path in the design, it will also consider configurations of PSM multiplexers that can create combinational loops. It is therefore not easily possible to constrain the timing of the design by simple definition of the final clock period, as Vivado will break the loops at arbitrary points. This generates long paths through different numbers of CLBs and PSMs, making it further impossible to constrain a path just between two specific PSMs. To solve this problem, these paths are broken manually. The individual atomic nets then have their delay constrained using the *set_max_delay* timing exception, ensuring that the design still meets timing and forcing the timing driving optimization to operate. These constraints will lead to path segmentation, which in this case is the desired outcome. In addition, it will add false path constraints on the original long paths automatically. Path segmentation can affect logic placement and timing results, so special care needs to be taken when examining the Vivado timing reports. In addition to path constraints, we define four clocks for our design: The primary clock as well as three auxiliary clocks used for the configuration of PSM, IOB and CLB elements. The frequency of configuration logic is less important than the application frequency, so configuration clocks will target a lower frequency, avoiding over constraining the design.

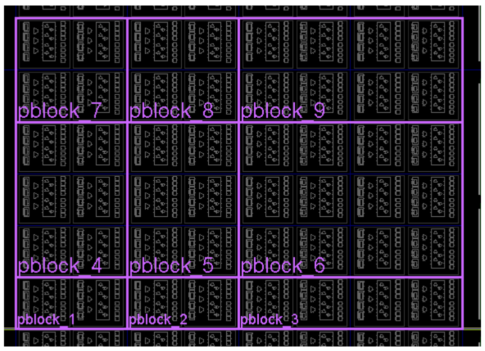
Extracting Metrics The Vivado timing report includes all details to judge how far the design met the timing constraints and usually provides the authoritative source in knowing the delay of all the nets. But in case of the V-FPGA, this report can not be used to extract meaningful data: The combinational loops, path delay constraints and path segmentation hide the important delays of the atomic nets from the timing report. Even though the target value for these nets is given using the path delay constraints, it is still useful to extract the real delays. To remedy this, a TCL script was written to extract the delays manually, using the *get_net_delay* command to get the delays of atomic nets. To get the delay of a whole path consisting of more than one net, a recursive search for nets is done and parts are summed up until the destination of the path is reached.

4 Logic Placement Strategies

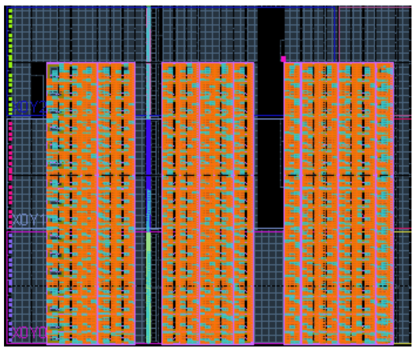
In the following, we discuss the three manual placement strategies in detail. We primarily use the uniformity metric to guide development of the strategies, then assess critical path delay and area in the evaluation.

4.1 Basic P-Block Strategy

In the basic P-Block strategy, we contain each tile in a single Partition Block (P-Block): We create a block with suitable size, then use the *add_cells_to_pblock* TCL command to add all cells of a tile to the block. Before a block can be created, the size of the block must be determined. We therefore investigate two options to derive the block size: The first option is to use a size with same width and height for all tiles, resulting in quadratic tiles. The largest tile dimensions are then taken as the unified size for the P-Blocks of all tiles. Alternatively, the size can be chosen according to the required area in each tile. This requires additional rules for tile sizes, to keep the rectangular layout of V-FPGA and avoid irregular layout results. Therefore, the horizontal size for all tiles in the same column and the vertical size for all tiles in the same row have to be identical, leading to rectangular tiles. Figure 2a demonstrates the second option, showing the generated floorplan for a small 3x3 V-FPGA.



(a) Simple Floorplan.



(b) Split Target Area.

Fig. 2: Floorplanning and P-Blocks: a): Floorplan for 3x3 V-FPGA using individually calculated sizes for each tile. To keep the layout regular, widths and heights of tiles are adjusted accordingly. b): P-Block floor plan demonstrating a larger V-FPGA design. The start point location was forced, so the target area must be split into three smaller areas because of intersecting hard blocks.

Before the V-FPGA can be placed, a suitable host FPGA location and area has to be determined. This step has to consider non-uniformity of the host FPGA: For example, Virtex 7 devices have a rectangular structure with larger gaps (more than 2 columns) and smaller gaps (2 columns) between CLBs. This is caused by I/O banks, clocking and other support logic. In addition, Digital Signal Processor (DSP) and Block RAM (BRAM) blocks are distributed over the chip between CLB columns. In order to reduce net length between placed logic blocks, the largest location with no gaps will be selected. This is supposed to improve net delays and support the rectangular layout of the V-FPGA. Finding the location and area consists of the following steps:

1. Estimate the overall area needed for the design using total CLB count.
2. Create a 2D array which represents available and used CLBs. Then search for the largest possible target area, only considering areas without blockages larger than an accepted gap. A reasonable value for Virtex 7 is two, allowing DSP and BRAM gaps but avoiding larger ones.
3. Calculate tile dimensions, fitting all tiles in square form in the target area.
4. If the previous step fails, set the dimension ratio of all tiles relative to the vertical and horizontal dimensions of the selected target area.
5. If the largest contiguous target area is not large enough to fit the complete design (step three and four failed), the steps are reevaluated. In this reevaluation, multiple disconnected areas are allowed, yielding split target areas as shown in figure 2b.
6. Find the vertical and horizontal dimensions of all tiles according to their resource usage.
7. Normalize the dimensions of all tiles in the same column or row.

After the tile sizes, host FPGA location and target area have been determined, the following step completes the basic P-Block placement:

8. Map all V-FPGA cells to the P-Blocks belonging to their tile.

4.2 Nested P-Block Strategy

In addition to the P-Blocks used in the first strategy, this strategy introduces up to two additional P-Blocks within each tile. Logic belonging to the V-FPGA CLBs and IOBs is mapped to these nested P-Blocks accordingly: When defining the P-Blocks, all assigned logic cells are forced into the blocks, but this does not prevent placing any additional unassigned cells into them. Based on this idea, we introduce two more variants in addition to the rectangular vs. quadratic layout distinction: In the partially nested strategy, we use the outer P-Block for the tile and nested blocks for IOB and CLB, but the PSM is only constrained by the outer P-Block. This gives Vivado the freedom to place the PSM in the remaining outer P-Block area, or place part of it inside the nested P-Blocks. In the fully nested strategy, we force Vivado to not place any PSM logic in the nested P-Blocks, prohibiting usage of remaining logic cells in them. Figure 3 demonstrates the concept for a 5x5 CLB V-FPGA.
The placement script is extended with the following steps to create the nested P-Blocks:

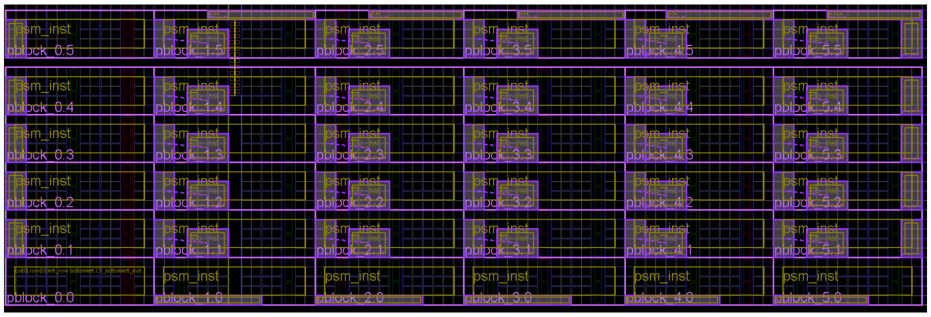


Fig. 3: V-FPGA floorplan with nested P-Blocks. The nested CLB P-Block is divided into two pieces to ensure the minimum possible area is used. The top right corner tile has an extra nested P-Block for its second IOB unit. No internal P-Block was used at all in the bottom left corner tile, as it only contains a PSM.

1. The internal P-Blocks can consist of multiple rectangles. The CLB P-Block is placed at the bottom left corner with height at most equal to the height of the tile minus one. This guarantees some freedom to IOB P-Block and to ensures distribution of the PSM unit over the tile P-Block.
2. The IOB P-Block is placed within the tile P-Block. The side is determined according to the tile type.

4.3 Fine-Grain Manual Placement Strategy

This strategy further constrains logic, directly mapping the relevant LUTs and flipflops to specific LUTs or flipflops in the 7 series host CLB. As there are numerous ways to place the logic within a tile, a manually derived layout is chosen instead of trying to find a fully automated one. The strategy is then made generic to support different V-FPGA parameters, but the layout is fixed to the V-FPGA and therefore cannot be reused for completely different applications. Evaluation of different manual layouts led to a placement as was presented in figure 1a: The PSM is located in the upper right corner and the CLB is placed in the lower part of the tile. Figure 4 shows the device view in Vivado after the manual placement strategy has been applied.
The implementation of this strategy operates on two lists for each tile P-Block, an instruction list and a list of the free host FPGA LUTs. The instruction list contains simple V-FPGA logic element place instructions, interleaved with sorting instructions. It is processed element by element, either placing logic elements or resorting the list of free resources. When an element placement instruction is processed, the logic elements are mapped sequentially to the elements in the sorted list of free resources, starting at a specified offset. When a resorting instruction is found, the resorting algorithm sorts the list of remaining available host LUTs. It sorts horizontally or vertically and uses ascending or descending sorting order, depending on the instruction. As an example, the *sort_xy_dd* instruction sorts first based on the x location, and if the x value is the same for

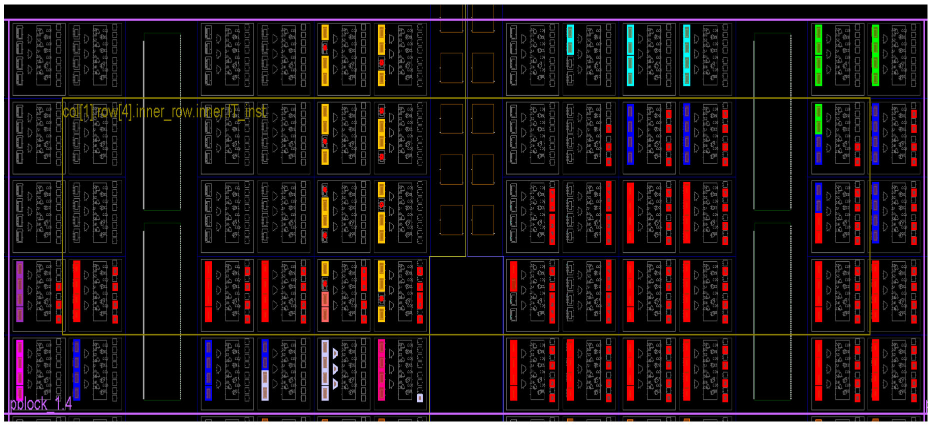


Fig. 4: V-FPGA tile (type 9) placed using the manual placement strategy. Multiplexers of the PSM's top, right, bottom and left side are marked sky blue, green, dark blue and yellow. The CLB is located at the bottom with the LUT, two internal multiplexers and D-flipflop colored in white. Red color represents the configuration units of the tile. Blue blocks at the bottom depict write connection boxes, whereas read connection boxes make up remaining logic around the LUT.

some CLBs, it uses y as secondary criteria. Descending sorting is applied in both cases. This specific instruction is used to sort the list of available logic elements before placing the right and left multiplexers of the PSM, as they need to be placed vertically from the top right corner. Sorting is always done on the list of free resources, so the length of this list decreases as the placement process proceeds. This makes it possible to reach every single CLB in the P-Block, not just the ones at the borders.

5 Evaluation of the Placement Strategies

To evaluate the different strategies with different V-FPGA parameters, three V-FPGA designs of increasing size have been implemented: A small 2x2 design with 4 CLB tiles total (track width 2), a 5x5 design with 25 CLB tiles (track width 10) and a 8x8 design with 64 CLB tiles (track width 9). The larger track width for the second design has been used to evaluate influence of increasing routing congestion, compared to the influence of the design size. All designs have been evaluated both with fine grain timing constraints and without fine grain timing constraints. We compare the three strategies presented previously, with both quadratic and rectangular tiles for the P-Block strategies. For the manual placement strategy, we use only rectangular blocks, but compare two different placement script variations. In the nested P-Block strategy, we consider both the partially and fully nested variants. Comparisons between the three proposed strategies are held in the previously described metrics of uniformity, worst delay and area. All results are normalized to the Vivado 2019.1 default synthesis results as a baseline.

Uniformity We assess uniformity as defined in the metrics chapter. Table 1 shows standard deviation $\overline{\sigma}$ for the different designs. In all strategies, for smaller designs, better uniformity is achieved than for larger designs. Also in all strategies, the 5x5 design has worst uniformity when timing constraints are applied. This can be explained by two observations: The 5x5 design uses wide track widths, which increases routing congestion in timing constrained designs and makes it more difficult to reach uniform values. Additionally, in larger designs, non-uniformity of the host FPGA is more prevalent, as the design spans a larger host FPGA area. There is no large difference between uniformity in quadratic or rectangular tile layout, but the rectangular layout performs better or equal to the quadratic layout for all strategies. When comparing the strategies, fully manual placement yields the best uniformity in timing constrained cases and is within 2 % of the best results in the other cases. Both variants are similar, but variant 1 provides slightly better uniformity.

Table 1: Standard deviation $\overline{\sigma}$ relative to Vivado standard synthesis results, comparing uniformity. "R" denotes rectangle based strategies.

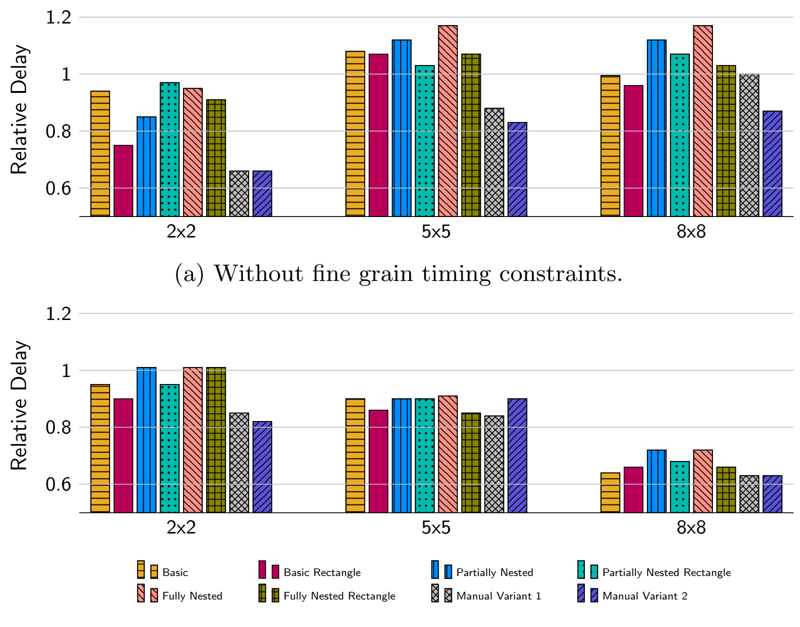
Timing Constraints	Design	Basic	Basic R	Partially Nested	Partially Nested R	Fully Nested	Fully Nested R	Manual Variant 1	Manual Variant 2
No	2x2	0.90	0.95	1.01	0.84	0.89	0.67	0.69	0.83
	5x5	0.82	0.79	0.80	0.80	0.80	0.81	0.81	0.81
	8x8	0.89	0.81	0.93	0.82	0.88	0.82	0.79	0.79
Yes	2x2	0.72	0.71	0.65	0.72	0.65	0.64	0.61	0.75
	5x5	1.04	0.98	1.02	1.02	1.00	1.00	0.90	0.90
	8x8	0.79	0.83	0.82	0.82	0.85	0.82	0.76	0.80

Table 2: Coefficient of variation $\overline{c_v}$ relative to Vivado standard synthesis results, comparing uniformity. "R" denotes rectangle based strategies.

Timing Constraints	Design	Basic	Basic R	Partially Nested	Partially Nested R	Fully Nested	Fully Nested R	Manual Variant 1	Manual Variant 2
No	2x2	0.97	1.01	1.03	0.85	0.92	0.71	0.74	0.92
	5x5	0.86	0.83	0.81	0.81	0.81	0.81	0.85	0.83
	8x8	0.93	0.85	0.93	0.82	0.90	0.81	0.83	0.82
Yes	2x2	0.81	0.79	0.72	0.78	0.72	0.68	0.70	0.77
	5x5	0.99	0.97	0.97	0.95	0.96	0.96	0.86	0.87
	8x8	0.88	0.89	0.88	0.87	0.90	0.88	0.78	0.83

Table 2 shows the coefficient of variation $\overline{c_v}$ for the different designs. Overall trends are similar to $\overline{\sigma}$, and for constrained designs, the manual placement variants again yield best results. The reduced standard deviation in delays therefore translates to relative improvements, which can be taken advantage of when synthesizing applications for the V-FPGA.

Delay Figure 5 shows how these improvements in uniformity translate to improvements in worst-delay τ for the different strategies. Figure 5a demonstrates that results without fine grain timing constraints need to be viewed with some scrutiny: Further analysis showed that without these constraints, Vivado puts little effort into routing optimization. This leads to worse results for most of the strategies, where the fully manual strategies are a notable exception and also yield consistent improvements in the unconstrained case. Interestingly, without these constraints, results of manual strategies get worse for larger designs. As the design is regular, we expected similar or better results for larger designs: The manual placement should be regular, whereas the baseline Vivado synthesis strategy may have to deal with congestion and increasing effort for large designs. Investigating this, we tightly constrained the nets in the design and reevaluated



(a) Without fine grain timing constraints.

(b) With fine grain timing constraints.

Fig. 5: Delays τ relative to Vivado standard synthesis results. Of all analyzed atomic needs, the largest increase or the smallest decrease, i.e. the worst case, is shown.

the results, shown in figure 5b. In this case, we can see significant improvements for all strategies over the baseline. Furthermore, the advantage of the strategies now increases for larger designs, which we attribute to routing and congestion issues in the Vivado default strategy for larger designs. It can also be seen that in most cases, using a rectangular floorplan leads to less delay than using the simpler square based floorplan. The two variants for manual placement produce the best and similar results, but depending on the size of the design, one may be slightly advantageous. For the largest design, we see an improvement of 37 % in the achieved delay.

Area Table 3 shows how the strategies affect the required host FPGA area. In the case without timing constraints, there is no consistent pattern. This is explained by the fact that the baseline is not timing optimized. For the constrained designs and small designs, most strategies even achieve reduced area usage. As designs get larger, the baseline logic is more localized and the relative overhead of the manual placement strategies increases. Here, worst case overheads of the fully manual strategies are 13 % and 16 %. We consider this as an acceptable tradeoff, as these strategies also achieve best results in delay reduction for large designs. Rectangular blocks most of the time lead to less used area, which is expected as they can fit the really required tile sizes more closely.

Table 3: Area of used or blocked CLBs relative to Vivado standard synthesis results. "R" denotes rectangle based strategies.

Timing Constraints	Design	Basic	Basic R	Partially Nested	Partially Nested R	Fully Nested	Fully Nested R	Manual Variant 1	Manual Variant 2
No	2x2	0.89	0.87	0.93	0.85	0.89	0.89	1.07	0.72
	5x5	0.99	0.99	1.04	1.00	1.03	1.03	0.99	0.91
	8x8	1.01	1.06	1.14	1.06	1.04	1.07	0.99	0.99
Yes	2x2	1.05	0.94	1.05	0.95	0.99	0.89	0.86	0.67
	5x5	0.93	0.95	0.98	0.92	0.98	0.99	1.04	0.98
	8x8	1.03	1.12	1.23	1.13	1.12	1.13	1.16	1.13

6 Conclusion

In this publication, we investigated varying degrees of customization in the Vivado FPGA logic placement process, aiming to reduce the delays in our V-FPGA implementation. We have shown that utilizing knowledge about the regularity of the design in logic placement can lead to significant improvements in net delay of the V-FPGA. Regressions in the delays were only found when the design was not constrained with fine grain timing constraints: Here, due to combinational loops in the V-FPGA, basic timing constraints do not constrain the design fully, preventing certain optimizations and leading to varying results. When constraining each atomic net in the V-FPGA, these regressions will not occur. We further showed that uniformity in the V-FPGA net timing can be used as an indicator to guide the design of the placement strategies: Increasing uniformity by up to 39 % in the timing results also lead to reduced maximum delay. Out of the strategies investigated, the most advanced strategy lead to best results. In this strategy, the location of individual CLBs was assigned completely manually using a TCL script and only routing was left to Vivado's automated algorithms. This approach lead to improvements in maximum delay of up to 37 % for the largest designs, at increased area costs of up to 16 %. Larger designs generally benefit more from the manual placement strategy.

To conclude, these results will allow for higher clock rates in user applications on V-FPGA, lowering the entry barrier for using it in more cases. In addition, the increased uniformity can be used to enable new concepts, such as overclocking of user applications or moving parts of the application dynamically on the V-FPGA. In general, more uniform V-FPGAs also provide more realistic insights for physical FPGA development, as they resemble their uniformity more closely.

References

1. Borriello, G., Ebeling, C., Hauck, S.A., Burns, S.: The triptych fpga architecture. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **3**(4), 491–501 (1995)
2. Figuli, P., Hübner, M., Girardey, R., Bapp, F., Bruckschlögl, T., Thoma, F., Henkel, J., Becker, J.: A heterogeneous soc architecture with embedded virtual fpga cores and runtime core fusion. In: 2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS). pp. 96–103 (2011)
3. Figuli, P., Ding, W., Figuli, S., Siozios, K., Soudris, D., Becker, J.: Parameter sensitivity in virtual fpga architectures. In: Wong, S., Beck, A.C., Bertels, K., Carro, L. (eds.) Applied Reconfigurable Computing. pp. 141–153. Springer International Publishing, Cham (2017)
4. Harbaum, T., Schade, C., Damschen, M., Tradowsky, C., Bauer, L., Henkel, J., Becker, J.: Auto-si: An adaptive reconfigurable processor with run-time loop detection and acceleration. In: 2017 30th IEEE International System-on-Chip Conference (SOCC). pp. 153–158 (2017)
5. Shi, M., Bermak, A., Chandrasekaran, S., Amira, A.: An efficient fpga implementation of gaussian mixture models-based classifier using distributed arithmetic. In: 2006 13th IEEE International Conference on Electronics, Circuits and Systems. pp. 1276–1279 (2006)
6. Sidiropoulos, H., Figuli, P., Siozios, K., Soudris, D., Becker, J.: A platform-independent runtime methodology for mapping multiple applications onto fpgas through resource virtualization. In: 2013 23rd International Conference on Field programmable Logic and Applications. pp. 1–4 (2013)
7. Yuan, J., Chen, J., Wang, L., Zhou, X., Xia, Y., Hu, J.: Arbsa: Adaptive range-based simulated annealing for fpga placement. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **38**(12), 2330–2342 (2019)