

A Customizable Framework for Application Implementation onto 3-D FPGAs

Kostas Siozios, *Member, IEEE*, and Dimitrios Soudris, *Member, IEEE*

Abstract—Integrating more functionality in a smaller form factor with higher performance and lower-power consumption is pushing semiconductor technology scaling to its limits. Three-dimensional (3-D) chip stacking is touted as the silver bullet technology that can keep Moore’s momentum and fuel the next wave of consumer electronic products. Additionally, the complexity of digital designs imposes that CAD algorithms are getting harder and slower. This article introduces a framework for application implementation onto 3-D reconfigurable architectures. In contrast to existing approaches, the proposed solution is customizable according to constraints posed by the application and the target 3-D device in order to improve performance metrics. Experimental results highlight the effectiveness of our framework, as we achieve average enhancements in terms of maximum operation frequency and power consumption by 35% and 47%, respectively, as compared to state-of-the-art algorithms.

Index Terms—FPGA, Three-dimensional integrated circuits, Partitioning, Placement, Routing

I. INTRODUCTION

FIELD-Programmable Gate Arrays (FPGAs) were first introduced almost two and a half decades ago. Since then, they have exhibited rapid growth and have become a popular implementation medium for digital systems. The programmable nature of their logic and routing interconnect make them flexible and general purpose but at the same time it makes them larger, slower and more power consuming as compared to the standard cell Application-Specific Integrated Circuits (ASICs).

For decades, semiconductor manufacturers have been shrinking transistor size in integrated circuits to achieve the yearly increases in performance described by Moore’s Law. This Law exists only because the RC delay was negligible, as compared to the signal propagation delay [1]. However, for sub-micron technology, the RC delay becomes a dominant factor. Previous studies showed that at 130nm technology node, approximately 51% of the microprocessor’s power is consumed by interconnect fabric [2]. This has generated many discussions concerning the end of device scaling as we know it, and has hastened the search for solutions beyond the perceived limits of current devices.

Three-Dimensional (3-D) integration is considered as a technology that will extend Moore’s Law into the next years.

Stacking multiple dies in the vertical axis and interconnecting them using very fine-pitch Through-Silicon Vias (TSVs) enables the creation of chips with very diverse functionalities implemented in different process technologies in a very small form factor. Moreover, the locality along the z -axis enables shorter routing paths, which in turn improves performance and power metrics, as compared to the corresponding 2-D system implementations [3].

The benefits of using 3-D integration in FPGAs are especially great, since these architectures suffer from data communication problems; the delay and power consumption of routing network are the main bottlenecks compared to ASIC implementations [4] [5]. The interest for designing 3-D reconfigurable architectures was already addressed by the reconfigurable industry. Typical examples are the 3-D FPGAs provided by Tezzaron [6], as well as the 2.5-D Xilinx Virtex-7 and UltraScale devices [7]. Besides, recently, Xilinx announced the 16nm 3-D UltraScale+ FPGA family, which exhibits a $2\times-5\times$ performance-per-watt advantage over comparable system designed with Xilinx’s 28nm devices [8].

Apart from the advancement in process technology, the performance depends also to the employed Computer-Aided Design (CAD) algorithms. Table I provides a qualitative comparison among recently proposed academic tool flows for application implementation onto 3-D FPGAs. A number of conclusions might be derived based on this analysis. Among others, the majority of existing tools focus only to homogeneous 3-D platforms. Although FPGAs are commonly designed as homogeneous architectures, the integration of different process technologies (e.g. logic, memory) into the same chip [9] and the design of domain-specific 3-D platforms [10], achieve significant higher performance metrics [1] [3] [4]. Moreover, none of the existing CAD algorithms are customizable according to requirements posed either by the application, or the 3-D device. Finally, it is worthwhile to mention that the available algorithms [11] [12] [13] [14] [15] [16] [17] [18] rely mostly on straight-forward extensions of existing 2-D tools, which cannot fully exploit the benefits of 3-D technology. On contrary, physical design in the 3-D realm requires fresh ideas (i.e. new algorithms and cost functions). For instance, existing flows pay effort to minimize the connections between layers. However, upcoming sections highlight that such an objective is not the case for 3-D FPGAs, since these inter-layer connections are fabricated in

TABLE I: Qualitative comparison among tool flows for 3-D reconfigurable platforms.

Feature		MEVA-3D [11]	TPR [12]	TPR [13]	VPR3D [14]	3D-Tree [15]	PROPOSED (previous version [10] [16] [17])
Architecture	3-D technology	TSV	TSV	TSV	SSIT	TSV	wirebonding, TSV, SSIT
	Heterogeneous layers	yes	no	no	no	no	yes
	Inter-layer routing	uniform	uniform	uniform	uniform	uniform	uniform, full-custom
Algorithm	CAD tuning	no	no	no	no	no	application-specific
	Partition engine	N/A	sim. anneal.	hMetis	hMetis	hMetis	tabu
	Partition objective	min-cut	min-cut	min-cut	min-cut	min-cut	constrained max-cut
	Placement engine	N/A	sim. anneal.	sim. anneal.	sim. anneal.	sim. anneal.	sim. anneal.
	Routing engine	N/A	pathfinder	pathfinder	pathfinder	pathfinder	pathfinder
Evaluation	Wire-length	yes	yes	yes	yes	yes	yes
	Delay	yes	yes	yes	yes	yes	yes
	Power	yes	no	no	yes	yes	yes
Other	Graphical interface	no	no	no	yes	yes	yes
	Public available	no	yes	yes	no	no	yes

application's and device's characteristics. In particular, the contributions of this article are:

- We prove that existing frameworks cannot fully benefit from the architectural features found in 3-D FPGAs in order to improve performance metrics.
- We introduce a novel framework for physical design onto 3-D FPGAs. The CAD algorithms of this framework can be customized according to application's and platform's characteristics.
- We introduce algorithms for addressing the netlist partitioning, placement and routing problems targeting 3-D reconfigurable platforms.
- We provide a software-supported systematic methodology for performing power/energy estimation regarding designs mapped onto 3-D FPGAs.

The proposed framework is evaluated with various benchmarks and 3-D FPGA devices. More thoroughly, experimental results shown average reduction of total wire-length, critical path delay (i.e. maximum operation frequency) and power consumption by 9%, 35% and 47%, respectively, as compared to state-of-the-art algorithms. Additionally, we show that the proposed algorithmic customization achieves to enhance the efficiency of our framework without imposing any overheads in term of execution run-time.

The rest of the article is organized, as follows: Section II presents the architectural template for the underline 3-D FPGA, whereas Section III describes the motivation behind this work. The proposed framework, as well as the employed CAD algorithms, are discussed in Section IV. Experimental results that quantify the efficiency of our solution against to state-of-the-art relevant tools are provided in Section V. Finally, Section VI summarizes the respective conclusions of this work.

II. ARCHITECTURAL TEMPLATE FOR 3-D FPGA

This section describes in detail the architectural template of the target 3-D FPGA, depicted schematically at Fig. 1. The 3-D architectures have up to five layers ($l \leq 5$), each of which is modeled as a homogeneous array of logic blocks in a two-level hierarchy. More thoroughly, the first level of hierarchy, also referred to as Basic Logic Element (BLE), consisted of a K -input lookup table (LUT) and a flip-flop (F/F), can

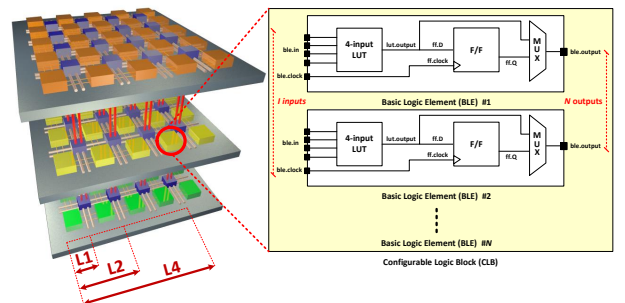


Fig. 1: Architecture template of the target 3-D FPGA.

implement any K -input logic function. The second level of hierarchy, mentioned as Configurable Logic Block (CLB), is formed by a group of N BLEs. For our experimentation we consider an architecture with $K=4$ and $N=5$. Previous studies have shown that such an architectural template provides a good compromise among the application's critical path delay, the power consumption and the area overhead [19]. Although there are a total of $K \times N$ inputs and N outputs inside the CLB, our architecture, similar to commercial devices, uses clusters that are less than fully connected. More specifically, the number of inputs to the logic cluster (I) was set based on the cluster size (N) and the LUT size (K) using the formula $I = \frac{K}{2} \times (N + 1) = 12$ in order to ensure that 98% of the BLEs can be utilized [20].

The interconnection infrastructure inside each layer, consisted of routing channels among CLBs, is also of high importance. A routing channel contains a number (W_H) of individual routing tracks, while each track is formed by segments which travel a distance L in CLBs before being interrupted by a programmable switch found inside a Switch Box (SB). The routing network modeled within this article relies on a multi-segment interconnection scheme with segments $L1$, $L2$, $L6$ and *longlines* (that span the entire device), whereas similar to Xilinx devices the distribution of these segments per channel is 8%, 20%, 60% and 12%, respectively [7]. As the connections with longer segments impose fewer switches, they lead to higher performance and lower power consumption. On the contrary, the shorter lengths provide higher flexibility for routing bends. Consequently, the employed multi-segment

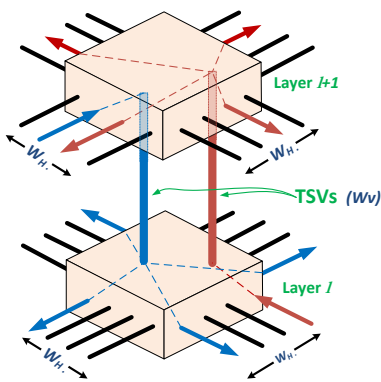


Fig. 2: Connections inside a 3-D SB.

interconnection scheme maximizes the flexibility to route a signal within a layer. Regarding the inter-layer connectivity, it is realized with vertical aligned TSVs [21] found inside SBs. Fig. 2 gives an example of the connections that take place inside a 3-D SB. Additional details about the design of these components can be found in [10]. Since the fabrication of these vertical interconnects with high density and yield is a complicated task [3] [4], their number is significantly limited compared to the rest routing tracks (W_H). For our experimentation we assume 3-D FPGAs with $W_V=3$ and $W_H=50$.

Table II summarizes the architectural properties for the target 3-D FPGA. The physical and electrical equivalent characteristics for the technology parameters are based on relevant publications [21] [22] [23]. Although our experiments make use of this fairly standard island-style template, the introduced framework is also applicable to any other architectural organization. However, the selection of a homogeneous device is performed solely for comparison purposes against relevant tools¹. Additionally, as we will discuss later, the selected number of TSVs per 3-D SB ($W_V = 3$) is the minimum one for successful application's P&R with existing (reference) state-of-the-art CAD tools [12] [13]. Note that for the sake of completeness, the value of W_V is constant for all experiments.

III. MOTIVATION AND CHALLENGES

The efficiency of application mapping onto a 3-D FPGA is tightly coupled to the employed CAD algorithms. This imposes the requirement for more powerful, native 3-D physical design tools that are built from ground up and are capable of handling 3-D aware optimization objectives. However, in contrast to this, the majority of existing tools rely on straightforward extensions of algorithms targeting 2-D platforms [11] [12] [13] [14] [15] [16] [17] [18].

In order to demonstrate that these tools do not fully benefit from the 3-D technology, Fig. 3 plots the percentage of utilized TSVs for devices consisted of $l \in \{2, 3, 4, 5\}$ layers. For demonstration purposes, the vertical axis is normalized over the number of fabricated TSVs per device². The target 3-D FPGA follows the architectural template discussed in Section II, while the application implementation is performed with

¹The available tool flows for 3-D FPGAs cannot handle devices consisted of heterogeneous components (e.g. memories, DSP, embedded processors, etc).

²The number of fabricated TSVs per 3-D FPGA is depicted at Table IV.

TABLE II: Architectural parameters for our experimentation.

Category	Parameter	Value
Technology	CMOS	45 nm
	shape	square
	layers	homogeneous
	style	island
	cluster size	$N = 5$
	LUT size	$K = 4$
	number of layers	$l \in \{2, 3, 4, 5\}$
	layer's size	best fitted to circuit size (see Table IV)
	routing channel width (X and Y directions)	$W_H = 50$
	vertical channel width (TSVs per 3-D SB)	$W_V = 3$
TSV [21]	distribution	uniformly per layer
	shape	square
	die-bonding type	face-to-back
	length	4-9 μm
	diameter	1.2 μm
	min. pitch	4 μm
	resistance (R)	0.35 Ω
	capacitance (C)	2.5 fF
Routing wires per layer	length	$L1, L2, L6$ and <i>longlines</i>
	RLC modeling	PTM 45 nm [22] [23]

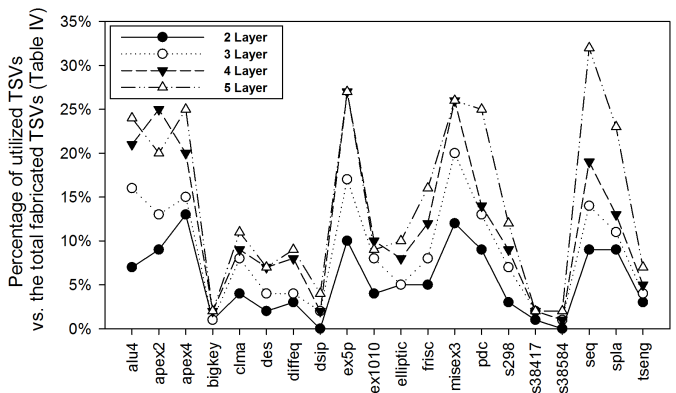


Fig. 3: Analysis of utilized TSVs per benchmark.

state-of-the-art CAD algorithms in academia for 3-D FPGAs, namely the hMetis netlist partitioning [18] and the TPR P&R [13].

The results of Fig. 3 indicate that on average only 5%, 9%, 12% and 15% of the fabricated TSVs are utilized for devices consisted of 2, 3, 4 and 5 layers, respectively. Specifically, as we will discuss later, only a small subset (on average 5%) of the 3-D SBs fully utilize the fabricated TSVs. Furthermore, apart from a few benchmarks that exhibit spikes at the utilization ratio (ranging up to 25%-30%), the majority of benchmarks utilize even fewer TSVs compared to the corresponding average values. One more conclusion might be derived from this analysis. Thoroughly, application implementation onto devices consisted of additional layers does not impose higher utilization ratio for the TSVs. Note that, the overall picture of Fig. 3 is not affected by the employed benchmark suite.

These outcomes occur mainly because the CAD tools rely on algorithms initially developed for 2-D platforms (e.g. multi-chip architectures). Since the inter-chip connectivity at these platforms exhibits considerable higher electrical equivalent

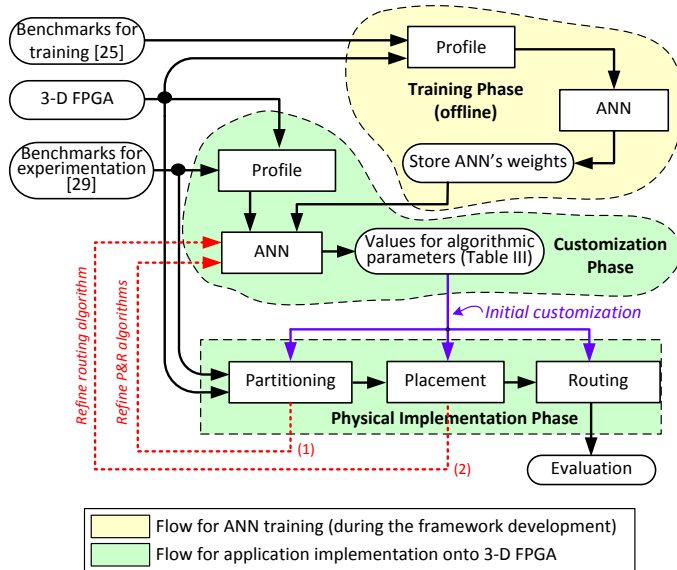


Fig. 4: Proposed framework for application implementation onto 3-D FPGAs.

characteristics (*RLC*) compared to a TSV connection³, the physical implementation algorithms focus mostly on *min-cut* partitions (i.e. to minimize the number of connections among partitions) in respect to the balance of partitions size (number of CLBs per partition). However, at the 3-D domain, this is not the case because the spatial locality among application’s functionalities assigned to different layers reduces the overall routing wire-length. Thus, mentionable performance enhancement is feasible in case that the TSVs are appropriately employed. Additionally, the concept of utilizing more efficiently the inter-layer connections is of outmost importance at the FPGA domain, since the TSVs are fabricated in advance of application implementation; hence, no additional cost is imposed if the design utilizes these TSVs.

IV. PROPOSED FRAMEWORK

This section introduces the proposed framework for application implementation onto 3-D FPGAs. This framework, illustrated at Fig. 4, consists of three phases, namely *training*, *customization* and *physical implementation*. In contrast to relevant approaches, the employed CAD algorithms are customizable according to application’s and platform’s characteristics with an Artificial Neural Network (ANN).

The efficiency of an ANN highly depends on the weights of its neurons. These weights are computed with a special procedure, referred to as *training*. During this phase, a representative number of benchmarks are profiled to extract various characteristics both of application (i.e. number of primary inputs/outputs, max/average fanout, number of blocks per net) and platform (i.e. number of layers, availability of resources and TSVs). Based on this data, the ANN is trained to provide the optimum customization of CAD algorithm’s parameters per application and target 3-D FPGA. Table III summarizes these algorithmic parameters for customization, as well as their

TABLE III: Algorithmic parameters for tuning.

Parameter	Description	Range
Partitioning		
g	aggressiveness factor for TSVs	0.5 – 0.8
ITM	size of intermediate short-term memory	1% – 3% of CLBs
LTM	size of long-term memory	2% – 5% of CLBs
Placement		
$q(i)$	scaling factor for network i	1.0 – 2.65
ϕ	expansion of neighborhood’s distance	5% – 25%
σ	percentage of directed moves	5% – 15%
Q	CLB swaps per temperature	$10G^{0.05} - 10G^{1.33}$
Routing		
ϵ	scale congestion cost per track	0.9 – 1.3

range. Having a trained ANN, we proceed to the physical implementation phase, which consists of two consecutive steps: Initially, the CAD algorithms are customized according to the ANN’s output and then the optimized algorithms perform netlist partitioning, placement and routing onto the target 3-D FPGA.

The introduced framework has also two refinement loops (marked with red dotted lines) that improve further the quality of algorithmic customizations. Specifically, the output of netlist partitioning affects the actual utilization of TSVs during the application’s routing. As we will discuss at Section IV-B, the estimation of this parameter is feasible to be performed. Similarly, the outcome of placement algorithm imposes additional constraints to netlist routing, i.e. based on the spatial locality among CLBs. Thus, the refinement loops “(1)” and “(2)” provide additional flexibility to the ANN in order to derive either a more, or less, aggressive customization of CAD algorithms depending on the availability of non-utilized routing resources. Note that each tool from the physical implementation phase is executed only once.

The rest of this section describes the employed ANN, as well as the proposed algorithms for addressing the netlist partitioning, placement and routing problems.

A. Customization of CAD Algorithms

The algorithmic customization discussed at Fig. 4 is performed with an ANN, which is an interconnected group of artificial neurons that uses a mathematical, or computational model, to realize complex relationships between inputs and outputs. The rest of this subsection describes in detail the ANN’s architecture, as well as its training procedure.

1) *Architecture of Artificial Neural Network*: The architecture of our ANN consists of three layers namely *input*, *hidden* and *output*. Typically, each layer receives input(s) from the previous layer and forwards its output(s) to the next layer (feed-forward structure). Designing an ANN therefore means coming up with the number of hidden layers, as well as defining the internal organization for each layer.

An ANN’s layer is modeled using a weight matrix W , a bias vector b , and an output vector a . The modeling of multiple layers imposes that instead of a unique weight matrix W , we use an input weight (IW) and a layer weight (LW) matrix. At this notation, superscripts denote the source (second index) and the destination (first index) for network elements. The ANN’s input is encoded with a vector p , whereas the net input n is computed as the sum of the bias b and the product $W \times p$. Then, the sum is passed to the transfer function f in order to get the neuron’s output a .

³A first order comparison between a routing wire in 45nm technology and a TSV can be found in [24].

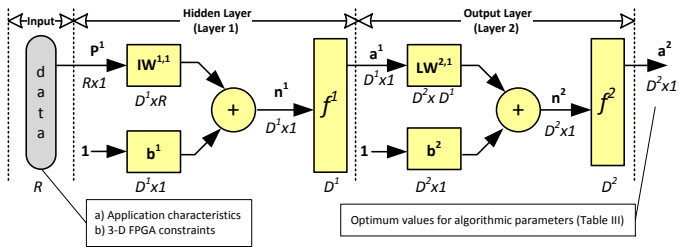


Fig. 5: Block diagram for the proposed ANN.

Fig. 5 plots the block diagram for the proposed ANN. This network relies on the multi-layer perceptron algorithm⁴, which is a widely studied and used ANN classifier. Among others, such an ANN can model complex functions, while it is robust to noise (good at ignoring irrelevant inputs) and flexible to adapt its weights in response to environment changes.

2) *Determining Network Parameters*: There are a number of parameters that have to be decided upon when designing an ANN. These parameters are:

- *Number of hidden layers*: The hidden layer(s) is(are) necessary to capture non-linear dependencies between the data's features and the variables that we are trying to predict. Additional layers of hidden neurons enable greater processing power and system flexibility (i.e. model more complex functions) but they come at the cost of higher complexity for training.
- *Number of neurons per hidden layer*: Having too many hidden neurons the system is over specified and is incapable of generalization. On the other hand, having too few hidden neurons, prevents the system from properly fitting the input data, and thus, it reduces the system's robustness.
- *Transfer function per layer*: The transfer function f^l translates the input signals to output signals for layer l . Previous studies have shown that the combination of different transfer functions improves the ANN's efficiency [25] [26].
- *Momentum*: It adds a fraction of the previous weight update to the current one in order to prevent the system from converging to a local minimum, or saddle point. Although a high momentum parameter increases the speed of system's convergence, it also creates a risk of overshooting the minimum, causing the system to become unstable. On the other hand, a low momentum coefficient leads to an ANN that cannot reliably avoid local minima, while showing down the training phase as well.
- *Learning*: What has attracted the most interest in ANN is the possibility to learn. This task is performed during the training phase, which relies on a combination of learning paradigms, rules and algorithms [25].

A widely accepted approach for optimizing the ANN's parameters is trial and error. However, this approach is inefficient and might miss the optimum network design. In order to overcome this limitation, we allow networks to be tested across the design parameter space with the procedure depicted at Fig. 6. For our analysis, the ANN's training is performed with

⁴supervised predictor of an input into one of several possible non-binary outputs

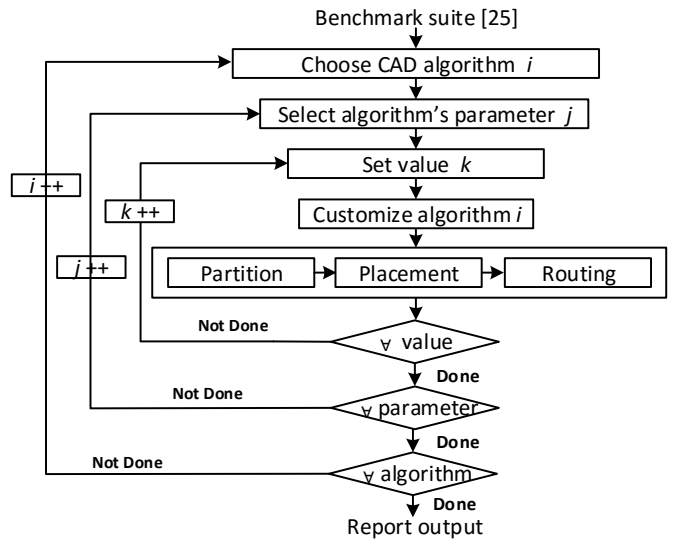


Fig. 6: Procedure for exhaustive search space exploration.

a representative number of benchmarks from Altera's QUIP suite [27]. The parameter i at Fig. 6 denotes the algorithm for customization ($i \in \{\text{partition, placement, routing}\}$), j refers to the algorithm's i parameter depicted at Table III, while k is the value where each of the parameter j ranges to. We assume that k takes 15 uniformly distributed values per parameter in the range depicted at the last column of Table III. Thus, in order to quantify benchmark z , this benchmark is partitioned, placed and routed with the customized flavor of the CAD algorithms according to the k values.

Fig. 7 outlines the exploration results for alternative number of hidden layers and neurons. Each of these parameters spans from 1 to 10, while the evaluation of ANN's architecture is performed according to the mean square error (RMSE) metric defined by Eq. 1.

$$RMSE = \sqrt{\frac{\sum_{z=1}^B (Y_z - \hat{Y}_z)^2}{B}} \quad (1)$$

where Y_z is the optimal combination of ANN's parameters as they are retrieved from the exhaustive search-space exploration discussed at Fig. 6, \hat{Y}_z is the predicted result by the ANN for benchmark z and B denotes the total number of benchmarks from the QUIP suite [27]. The term optimal refers to the combination of algorithmic's parameters that minimizes the average Energy×Delay product (EDP) among the studied benchmarks. This analysis indicates that optimal results are retrieved for an ANN with one hidden layer, which contains $D^1 = 8$ neurons.

The transfer function per hidden/output layer is another design parameter that has to be specified. For this purpose, a number of well-established functions, namely *hardlim*, *logsig*, *radbas*, *satlin*, *translg*, *sigmoid* and *linear* [25] [26] [28] were instantiated and their efficiency for our problem was quantified. Based on this experimentation, we found that minimum time for convergence is retrieved when the hidden and output layers employ the *sigmoid* and *linear* transfer functions, respectively.

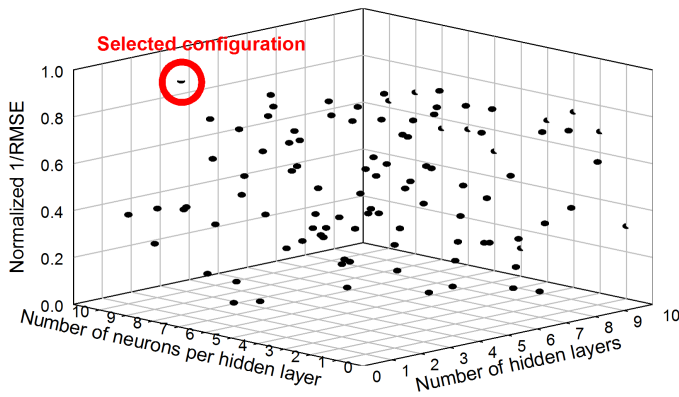


Fig. 7: Calibration of the proposed ANN.

Algorithm 1 Pseudo-code for the ANN’s training

```

1: Input:  $InPat$  // input patterns;
2: Input:  $max_{iter}$ ;
3: Input:  $learn_{rate}$ ;
4:  $Network \leftarrow ConstructNetworkLayers()$ ;
5:  $Network_{weights} \leftarrow InitializeWeights(Network)$ ;
6: for ( $i=1$  to  $max_{iter}$ ) do
7:    $Pat_i \leftarrow SelectInputPattern(InPat)$ ;
8:    $Output_i \leftarrow ForwardPropagate(Pat_i, Network)$ ;
9:    $BackPropagateError(Pat_i, Output_i, Network)$ ;
10:   $UpdateWeights(Pat_i, Output_i, Network, learn_{rate})$ ;
11: end for
12: Output:  $Network$ ;

```

After defining the ANN’s architecture, we proceed to the training phase, where the weight per neuron is determined. These weights represent an abstraction of the mapping of input vectors to the output signal for those benchmarks that the ANN was exposed to during training phase. The training for our ANN relies on a back-propagation with momentum algorithm (see Algorithm 1), since previous analysis has proven the efficiency of such an approach [25]. In more details, the momentum feature improves the rate of convergence (speedup’s the learning process) as compared to the conventional back-propagation algorithm, while further improvement is feasible by employing biased neurons with constant input equals to 1. In order to perform an efficient training, careful selection of representative (according to corresponding characteristics of real-life applications) benchmarks is necessary. The 70% of the selected benchmarks are employed for ANN’s training, whereas the validation and testing are performed with the rest 15% and 15%, respectively.

The initial network weights are tuned to random values. This is a widely accepted approach to overcome trapping to local error minima [25] [26]. However, such a selection imposes that two identical networks can produce very different calibration models once trained. Repeated evaluation of ANN is therefore essential in getting a clear idea of the capability of each design. Towards this direction, for each permutation of network parameters, we train 5 distinct networks in order to reduce the impact of the random initial starting conditions.

During the training phase, the weights are updated based on a modified delta rule, where an error signal is calculated

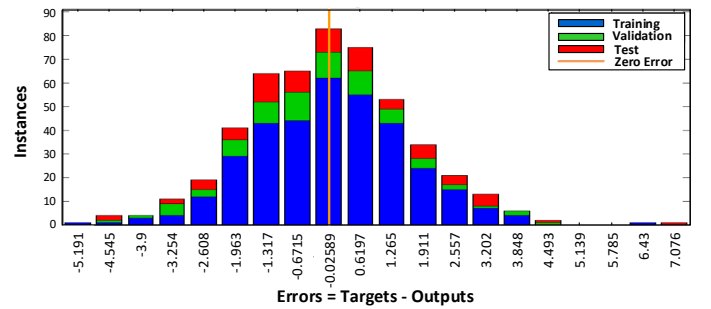


Fig. 8: Error analysis for the ANN’s training.

for each node and it is backwardly propagated through the network, starting at the output layer and weighted back through the previous layers. Fig. 8 quantifies the ANN’s efficiency, where different colors denote the errors for training, validation and testing. These results indicate that the error is small enough for the majority of studied benchmarks. Hence, one might claim that our ANN can address sufficiently the algorithm customization problem for almost any benchmark.

B. Partitioning and Layer Ordering

Netlist partitioning is an important problem associated with the physical design onto 3-D architectures. Mathematically, the netlist partitioning is an NP-hard problem, where no polynomial-bounded algorithm for finding the global optimal solution is likely to exist. Given an undirected hypergraph $G = (V, H)$ and V, H being the set of vertices and hyperedges respectively, a k -way balanced partitioning of G is defined as a function $p : V \rightarrow 1, 2, \dots, k$ that distributes the vertices of V among k disjoint subsets $S_1 \cup S_2 \cup \dots \cup S_k = V$ of roughly equal size. The partitioning function induces a new hypergraph $G_p = G_p(S, H_c)$, where $S = S_1, S_2, \dots, S_k$ and a hyperedge $S_i, S_j \in H_c$ exists if there are two adjacent vertices $u, v \in V$ such that $u \in S_i$ and $v \in S_j$. The set H_c corresponds to the set of cutting hyperedges of G induced by the partition. Regarding our implementation, the term vertex refers to application’s functionality technology mapped onto a CLB, while a hyperedge corresponds to a network path. Throughout this article we assume that benchmarks are represented as hypergraphs, hence the terms *benchmark*, *application*, *netlist* and *hypergraph* are used interchangeably.

The majority of existing CAD algorithms aim to minimize the connections among partitions, subject to design constraints such as the maximum partition size and the maximum path delay [18]. However, at Section III we showed that these algorithms cannot benefit from the availability of fabricated TSVs in order to derive shorter routing paths. Therefore, this subsection introduces our iterated tabu search algorithm (depicted at Algorithm 2) for addressing the netlist partitioning and layer ordering problems. The proposed solution relies on an existing algorithm [16], which was extensively revised for additional flexibility. More thoroughly, apart from the pure 3-D aware objectives, a number of algorithmic parameters (summarized at Table III) are customized with the proposed ANN.

The introduced algorithm is a global optimization strategy based on a hill climbing, which is characterized by aggressive

Algorithm 2 Partitioning and Layer Ordering Algorithm

```

1: Input: netlist hypergraph  $G = (V, H)$ ;
2: Input: number of partitions  $k$ ;
3:  $STL \leftarrow$  size of memory lists ( $ITM, LTM$ );
4:  $P_{best} \leftarrow \text{ComputeInitialSolution}()$ ;
5:  $TabuList \leftarrow \text{empty}$ ;
6: while ( $\text{cost}(P_{best})$  not improved further) do
7:    $CandidateList \leftarrow \text{empty}$ ;
8:   for ( $P_{candidate} \in P_{best\text{neighborhood}}$ ) do
9:     if ( $\text{ContainsAnyFeatures}(P_{candidate}, TabuList)$ )
10:      then
11:         $CandidateList \leftarrow P_{candidate}$ ;
12:      end if
13:   end for
14:    $P_{candidate} \leftarrow \text{LocateBestCandidate}(CandidateList)$ ;
15:   if ( $\text{Cost}(P_{candidate}) \leq \text{Cost}(P_{best})$ ) then
16:      $P_{best} \leftarrow P_{candidate}$ ;
17:      $TabuList \leftarrow \text{FeatureDifferences}(P_{candidate}, P_{best})$ ;
18:     while ( $TabuList \geq STL$ ) do
19:        $\text{DeleteOlderEntries}(TabuList)$ ;
20:     end while
21:   end if
22: end while
23: Output:  $\text{Return}(P_{best})$ ;

```

local search during each iteration. The tabu search maintains a short term memory (*TabuList*) to prevent the algorithm from returning to recently visited areas of the search space, referred as *cycling*. For this purpose, the short term memory is managed by a mechanism that involves historical information about the last moves. Note that the short term memory alone does not ensure that the search will be effective. Thus, two complementary memory mechanisms are employed: (i) the *intermediate-term memory* (*ITM*) to focus the search on promising areas of the search space (intensification), called *aspiration* and (ii) the *long-term memory* (*LTM*) to encourage useful exploration of the broader search space, called *diversification*. The size for these memories is customized according to the ANN's output.

Apart from the *ITM* and *LTM* customizations, the proposed implementation exhibits a number of differentiations as compared to similar tabu-based heuristic engines. Specifically, instead of random vertex selection, we favour moves that improve to some degree the quality of solutions, as it is defined by the cost function (Eq. 2). Note that we also use some form of randomization, since during each iteration a movement is selected randomly from a list of the most attractive candidates. Additionally, given a partition S_i , our algorithm defines and evaluates the relocation neighbourhood $RN(S_i)$ to be a set of all the solutions (i.e. partitions) that can be obtained from S_i by relocating a single vertex. The evaluation of derived partitions is based on the cost function given by Eq. 2.

$$\text{cost} = c \times W_{hcut} + (1 - c) \times \left(\xi \times W_{order} + (1 - \xi) \times W_{balance} \right) \quad (2)$$

where:

- $W_{balance}$: Balances the total weight for vertices among partitions. Let $|u|$ denote the weight of a vertex u . Then,

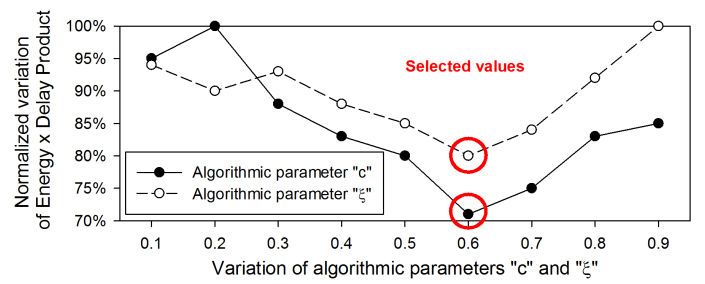


Fig. 9: Calibration of weighting factors for the netlist partitioning algorithm.

the weight $w(S_i)$ of a subset S_i is equal to the sum of weights of the vertices in S_i , where $w(S_i) = \sum_{u \in S_i} (|u|)$.

Since throughout this article we target homogeneous 3-D FPGAs, the weights at each vertex equals to 1. Thus, the $W_{balance}$ factor leads to solutions with similar number of vertices per partition.

- W_{hcut} : Minimizes the hyperedge-cut according to the aggressiveness factor g , as it is derived from the ANN. The hyperedge-cut is a first order metric for the number of utilized TSVs after successful applications' P&R. Smaller values for g factor lead to partitions that utilize fewer TSVs. On the other hand, higher values are expected to improve performance metrics due to spatial locality but imposes the utilization of additional TSVs.
- W_{order} : Minimizes the distance (in term of 3-D FPGA's layers) among the source node u and all the network's sink nodes. In contrast to existing platform-agnostic partitioning algorithms, this objective minimizes the wastage of TSVs by addressing the layer ordering sub-problem.

The values of parameters c and ξ are equal to 0.6, as they are defined based on the detailed search-space exploration depicted at Fig. 9. Note that such a combination of weighting factors gives higher importance to minimizing the hyperedge-cut according to aggressiveness factor g .

The outcome from netlist partition highly affects the number of utilized TSVs. Since the actual utilization of these resources is derived after successful application's routing, an estimation would be a valuable instrument for system designers. Such an estimation becomes far more important if we take into account that P&R is a timing consuming task. Thus, mentionable reduction at design cost is feasible by pruning the design space solutions that are known to violate system's constraints (e.g. exceed the number of fabricated TSVs). Such an estimation is also taken into consideration during the refinement loops discussed at Fig. 4.

The results of this analysis for the employed benchmark suite [29] are visualized at Fig. 10, where the left vertical axes give the ratio (according to Eq. 3) between the hyperedge-cut and the number of utilized TSVs, as they are retrieved after netlist partitioning and physical implementation, respectively. In other words, such a ratio corresponds to the overestimation of actual demand for TSVs based on the hyperedge-cut metric. For demonstration purposes, we plot also (with dots) the average ratios per layer. Based on this analysis, we might conclude that the ratio decreases monotonically as we proceed to devices with additional layers, since these layers provide

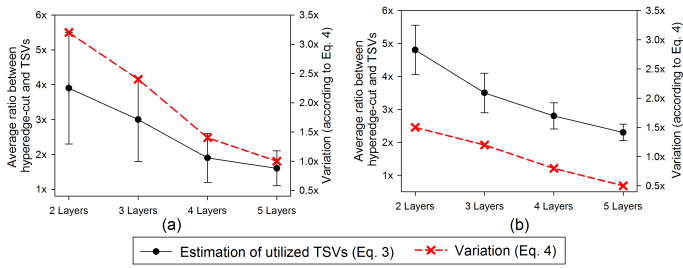


Fig. 10: Average ratio between the hyperedge-cut and the number of utilized TSVs (Eq. 3), as well the variation of this ratio (Eq. 4) for: (a) the existing TPR flow [13] and (b) the proposed framework.

higher flexibility to address the layer ordering sub-problem (W_{order} objective).

Apart from the average values, the variation of this ratio (i.e. difference between maximum and minimum value) is also crucial for accurately estimate the number of TSVs. In order to evaluate this goal, the right axes at Fig. 10 give the average variation of ratios among the studied benchmarks according to the Eq. 4. Although the TPR tool (Fig. 10(a)) leads to smaller average ratios in comparison with the proposed framework (Fig. 10(b)), our solution reduces the corresponding variations by 50% on average. Thus, our framework provides higher fidelity to the designer about the actual demand for TSVs early at the design phase.

$$ratio = \frac{hyperedge - cut \text{ after netlist partitioning}}{utilized TSVs \text{ after netlist routing}} \quad (3)$$

$$variation = \max(estimation) - \min(estimation) \quad (4)$$

C. Placement

Placement is one of the most influential steps in the FPGA CAD flow, as it is directly responsible for determining the relative locations of CLBs in order to minimize wire-length (an extension of traditional 2-D half perimeter bounding box by adding the offset of the z coordinate) and critical path delay subject to the minimum possible penalty in routing congestion.

This subsection presents a high-quality placement algorithm targeting to 3-D FPGAs. The proposed placer, depicted at Algorithm 3, is based on simulated annealing; a heuristic for minimizing an objective cost function which takes real values over a set of states. Given a perfect cooling schedule, simulated annealing will ultimately converge to an optimal solution; however, the design space exploration must satisfy some conditions for optimality to be assured [30].

The evaluation of candidate placements is performed based on the cost function described by Eq. 5. This function aims at minimizing the application's timing (T_{cost}), wire-length (W_{cost}) and power consumption (P_{cost}) metrics, while the weights α and β define the importance among these factors.

$$cost = \alpha \times T_{cost} + (1 - \alpha) \times \left(\beta \times W_{cost} + (1 - \beta) \times P_{cost} \right) \quad (5)$$

Algorithm 3 Placement Algorithm

```

1:  $temp \leftarrow$  Initial temperature  $T_A$ ;
2:  $counter \leftarrow 0$ ;
3:  $place \leftarrow$  Initial placement;
4: while ( $temp > T_0$ ) do
5:   while ( $counter < \max\_iter$ ) do
6:      $new\_place \leftarrow$  perturbation( $place$ );
7:      $delta \leftarrow cost(new\_place) - cost(place)$ ;
8:      $r \leftarrow random(0, 1)$ ;
9:     if ( $r < e^{-\frac{\Delta cost}{T}}$ ) then
10:       $place \leftarrow new\_place$ ;
11:    end if
12:     $counter++$ ;
13:  end while
14:   $counter \leftarrow 0$ ;
15:   $temp \leftarrow schedule(temp)$ ;
16: end while
17: return  $place$ ;

```

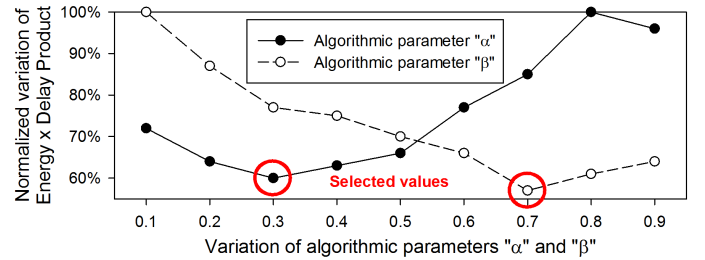


Fig. 11: Calibration of weighting factors for netlist placement algorithm.

$$T_{cost} = \sum_{i=1}^{total \ nets} \left(delay(u, v) \times crit(i)^{const} \right) \quad (6)$$

$$W_{cost} = \sum_{i=1}^{total \ nets} q(i) \times \left(\frac{bb_x(i)}{C_{av,x}(i)} + \frac{bb_y(i)}{C_{av,y}(i)} + \frac{bb_z(i)}{C_{av,z}(i)} \right) \quad (7)$$

$$P_{cost} = \sum_{i=1}^{total \ nets} \left(act(i) \times \gamma \lambda \frac{\sqrt{2} \times (3r + 3)}{(2r + 1) \times (2r + 2)} \times n_c^{r-0.5} \right) \quad (8)$$

where $delay(u, v)$ denotes the estimated delay between nodes u and v (a source-sink path of a network), $crit(i)$ gives the importance in term of how close to the critical path is the network i , and $const$ is a constant value. The $bb_x(i)$, $bb_y(i)$, $bb_z(i)$ correspond to the dimensions of the bounding cube for network i , while the parameters $C_{av,x}(i)$, $C_{av,y}(i)$ and $C_{av,z}(i)$ give the average width of routing tracks across the x , y and z directions, respectively, over the bounding cube of net i . The $q(i)$ parameter scales the wiring model discussed previously to take into consideration nets with more than 3 terminals [31]. The value of this parameter is application-specific and it is retrieved from the ANN. Specifically, the $q(i)$ parameter depends on the number of terminals of net i ; q is 1 for nets with 3 or fewer terminals, and slowly increases to 2.65 for nets with 50 terminals [31]. Finally, $act(i)$ is the switching

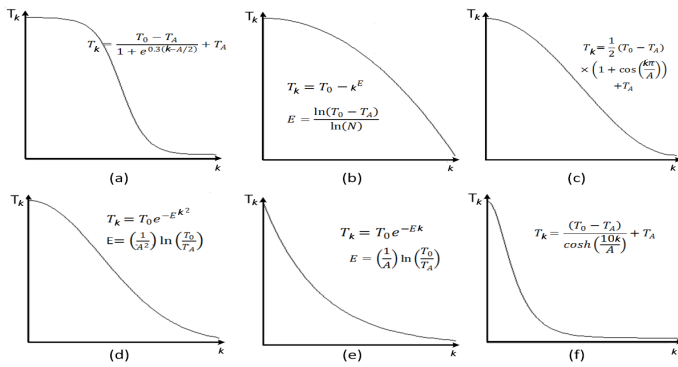


Fig. 12: Alternative cooling schemes for simulated annealing.

activity for network i , γ is a constant that denotes the inverse channel width utilization ratio, λ is the average number of inputs used in a logic block, r is the Rent's exponent and n_c is the number of CLBs required for a given circuit [32] [33]. The switching activity per network is computed with an extended version of the well-established in academia ACE 2.0 tool [34] [35]. Exhaustive exploration with various benchmarks and 3-D FPGAs indicate that optimum EDP is retrieved for $\alpha=0.3$ and $\beta=0.7$ (see Fig. 11).

The quality of placements is tightly firm to the employed cooling scheme, which defines the maximum allowed distance for CLB swapping [20]. In order to study more thoroughly the impact of this parameter, we evaluate the efficiency for six alternative cooling schemes (depicted at Fig. 12). The vertical axes (T_k) at this figure denote the temperature at iteration k , where $0 \leq k \leq A$ and A is the total number of moves during the annealing process. For the sake of completeness, the initial (T_A) and final (T_0) temperatures are identical among the studied cooling schemes.

These schemes were implemented as part of our placer and their efficiency was quantified based on the size of bounding cube. Fig. 13 plots the average values for this metric among the studied MCNC benchmarks [29]. For demonstration purposes the vertical axis is plotted in a normalized manner over the maximum value among the benchmark suite. Based on this analysis, we found that the cooling scheme described by Eq. 9 (Fig. 12(e)) minimizes the size of bounding cube, and therefore it is selected for the rest of our experimentation.

$$T_k = T_0 \times e^{-k \times \left(\left(\frac{1}{A} \right) \times \left(\ln \frac{T_0}{T_A} \right) \right)} \quad (9)$$

As FPGAs continue to grow in size, the large run-times incurred by simulated annealing are becoming prohibitive. Accordingly, instead of the conventional simulated annealing engine, which usually spends time revisiting previously explored states, the proposed algorithm is forced to explore also neighbor states to already known good solutions. These directed moves are likely to reduce the amount of time required for the annealing process to find the final (lowest cost state) placement. Specifically, the radius (Manhattan distance) for the neighbour locations, mentioned as φ , ranges between 5% and 25% of the maximum allowed distance for CLB swapping per temperature T_k [20].

Typically, a given directed move will not impact the same

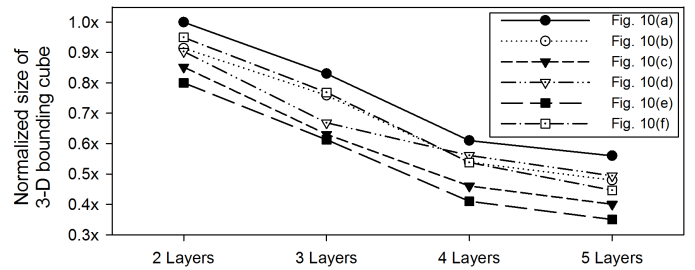


Fig. 13: Average size of 3-D bounding cubes for alternative cooling schemes.

changes in the cost function at different temperatures. Ideally, directed moves will be employed when they are most likely to improve the cost function. To accomplish this goal, our framework incorporates a parameter (σ) to define the percentage of directed moves compared to the total swaps per temperature. These moves have to be implemented very carefully; otherwise the risks of oscillation and converging to a local minimum are raised. Based on our exploration we found that as the temperature cooled, the directed moves become more effective than the random moves because at lower temperatures, almost all the logic blocks are placed in their optimal median ranges, and timing paths are relatively straight. Consequently, the random perturbations become less effective than the simpler directed moves. Considering that both φ and σ parameters are application-dependent, their values are retrieved from the employed ANN.

Finally, in our previous work we have shown that a reduction at the number of swaps that are quantified per temperature T_k improves the placer's execution runtime with a controllable overhead at the quality of derived solutions [36]. Specifically, existing placers based on simulated annealing, both for 2-D [20] and 3-D FPGAs [10] [12] [13] [14] [15] [17], perform $Q = 10 \times G^{1.33}$ swaps per temperature, where G is the number of utilized CLBs. In this article, we have extended the concept initially proposed at [36] towards two orthogonal directions: (i) to be applicable at 3-D FPGAs and (ii) to customize the number of swaps per temperature (Q) according to the ANN's output. For this purpose, two flavors of our framework are evaluated: the first of them, referred to "normal", performs $Q = 10 \times G^{1.33}$ swaps per temperature, similar to existing placers. On the other hand, the number of swaps at the second flavor, referred as "fast", is application-dependent and it is derived from the ANN. As it is depicted at Table III, in such a case the number of evaluated swaps per temperature ranges between $10 \times G^{0.05}$ and $10 \times G^{1.33}$.

D. Routing

The routing problem in FPGA domain relies on determining which programmable switches have to be turned on to connect all the CLB input and output pins required for the application's functionality. This problem becomes even more challenging for 3-D devices in the sense that inter-layer connectivity is provided only by a limited (as compared to planar routing wires) number of TSVs. Algorithm 4 gives the pseudocode for the proposed router, which is based on a modified pathfinder negotiated congestion/delay algorithm [37]. Flavors of pathfinder router are also used both in commercial (i.e. Xilinx, Altera), as well as academic tool flows [11] [12] [13] [14] [20].

Algorithm 4 Routing Algorithm

```

1: initialize  $\text{crit}(i, v) = 1.0$  for all nets  $i$  and sinks  $v$ ;
2: while (exist overused routing resources) do
3:   for (each net  $i$ ) do
4:     rip-up routing tree (net  $i$ );
5:     for (each sink  $v$  of net  $i$  in decreasing  $\text{crit}(i, v)$  order) do
6:       breadth first routing (sink  $v$ );
7:       for (all nodes in the path from  $u$  to  $v$ ) do
8:         update(congestion cost  $\varepsilon$ );
9:       end for
10:    end for
11:  end for
12:  update (historic congestion);
13:  compute (routing tree delay);
14:  update (timing graph);
15:  update (net  $\text{crit}(i, v)$ );
16: end while
17: compute (timing, power, wiring);
18: return (route);

```

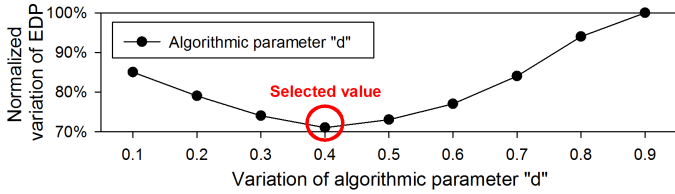


Fig. 14: Calibration of weighting factor for the netlist routing algorithm.

Initially, all signals in a placed design are routed in the best manner possible (e.g. minimum delay/wire-length, etc), permitting shorts between the signals (i.e. two or more signals may use the same wire). Then, the penalties associated with the shorts are gradually increased, and the signals are re-routed according to the lowest cost path available, in order to avoid shorts where possible. The process of increasing the penalties for shorts and re-route signals continues iteratively until all shorts are removed and the routing is feasible.

Eq. 10 gives the cost metric for quantifying a routing track h that forms a connection from source u to sink v . The value of parameter d equals to 0.4, as it minimizes the Energy \times Delay Product for the exploration depicted at Fig. 14.

$$\text{cost}(h) = \text{crit}(i, v) \times \text{delay}(h) + (1 - \text{crit}(i, v)) \times (\text{act}(i) \times \varepsilon \times \text{cong}(h)) \quad (10)$$

$$\text{cong}(h) = \text{cb}(h) \times \text{ch}(h) \times \text{cp}(h) \quad (11)$$

We define the criticality of a connection, $\text{crit}(i, v)$ as:

$$\text{crit}(i, v) = \max\left(\left[\text{MaxCrit} - \frac{\text{slack}(i, v)}{D_{\max}} \right]^{\vartheta}, 0\right) \quad (12)$$

where D_{\max} is the delay of the circuit's critical path and $\text{slack}(i, v)$ refers to the slack of the connection between the source and sink v of net i . The exponent ϑ , as well as the Max_Crit are parameters that control how a connection's

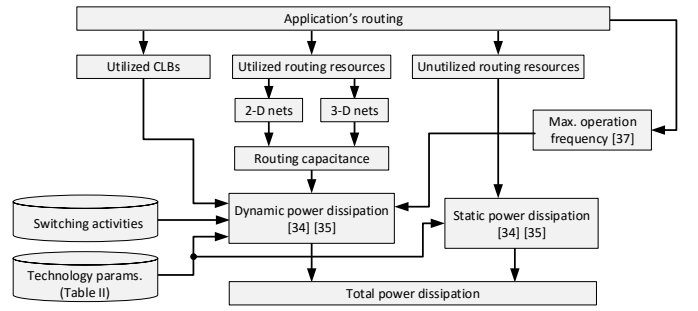


Fig. 15: Procedure for power estimation at 3-D FPGAs.

slack impacts on the congestion-delay trade-off in cost function (depicted at Eq. 10). The $\text{delay}(h)$ refers to the Elmore's delay for routing track h , while $\text{act}(i)$ is the switching activity for net i . Moreover, the $\text{cb}(h)$, $\text{ch}(h)$ and $\text{cp}(h)$ correspond to the base cost, the historical congestion cost and the present congestion cost of routing track h , respectively. Hence, the cost of using a track is a function of its current overuse and any overuse that occurred in prior routing iterations (see Eq. 11). Regarding the cost of an oversubscribed routing resource $\text{cp}(h)$, it is gradually increased after the completion of each iteration to discourage resource sharing. This selection forces networks with alternative routes to avoid using the oversubscribed resource, leaving it to the net that most needs it. Finally, the parameter ε (derived from the ANN) scales the $\text{cong}(h)$ according to the availability of TSVs and the value of hyperedge-cut⁵. Higher values of ε favor applications with a small hyperedge-cut; hence additional TSVs might be utilized to improve performance metrics without leading to unroutable designs.

E. Evaluation and Power Analysis

The last step in our framework deals with the evaluation of application's implementation onto the target 3-D FPGA. Towards this direction, a number of well-established models in academia are employed. More thoroughly, the maximum operation frequency and the total wire-length are computed based on the Elmore delay model [38] and a wiring model [20], respectively.

Furthermore, the power dissipation is a major concern for 3-D FPGAs [5]. Understanding the variation of this design parameter within these devices is the first step at developing power-efficient architectures and CAD tools. Although the problem of power dissipation was already noticed both from industry [1] [3] and academia [4], up to now there are no tools to explore the impact of different selections at the power/energy consumption. In order to overcome this limitation, we have developed a systematic procedure, depicted at Fig. 15, to perform power estimation for designs mapped onto 3-D FPGAs. Having as inputs the routing file, the switching activity per net, the maximum operation frequency and a number of technology parameters (e.g. power supply, transistor size, etc), it is feasible to calculate both static and dynamic power consumption for a design. The switching activity per

⁵The value of hyperedge-cut is retrieved from the *refinement loops* discussed at Fig. 4.

TABLE IV: Benchmark complexity and the array sizes of target 3-D FPGA devices.

Benchmark	CLBs	2-D FPGA (array size)	3-D FPGA (array size)			
			2 Layers	3 Layers	4 Layers	5 Layers
alu4×10	3,018	56×56	41×41	34×34	29×29	26×26
apex2×10	3,722	63×63	46×46	38×38	33×33	29×29
apex4×10	2,510	52×52	38×38	31×31	27×27	24×24
bigkey×10	3,634	62×62	45×45	38×38	32×32	29×29
clma×10	16,766	131×131	95×95	79×79	67×67	60×60
des×10	3,162	58×58	42×42	35×35	30×30	27×27
diffeq×10	2,994	56×56	41×41	34×34	29×29	26×26
dsp×10	2,740	54×54	39×39	33×33	28×28	25×25
elliptic×10	2,076	47×47	35×35	29×29	25×25	22×22
ex1010×10	9,090	97×97	71×71	58×58	50×50	44×44
ex5p×10	7,208	86×86	63×63	52×52	45×45	40×40
frisc×10	7,112	86×86	63×63	52×52	44×44	39×39
misex3×10	2,752	54×54	40×40	33×33	28×28	25×25
pdc×10	9,012	96×96	70×70	58×58	50×50	44×44
s298×10	3,862	64×64	47×47	39×39	33×33	29×29
s38417×10	12,812	115×115	84×84	69×69	59×59	52×52
s38584×10	12,894	115×115	84×84	69×69	59×59	53×53
seq×10	3,466	60×60	44×44	37×37	32×32	28×28
spla×10	7,262	87×87	63×63	52×52	45×45	40×40
tseng×10	2,094	47×47	35×35	29×29	25×25	22×22
Average values:	5,910	75×75	55×55	45×45	39×39	35×35
Average utilization:		95.5%	89.8%	88.2%	86.0%	85.2%

net is computed based on the well-established in academia ACE 2.0 tool [34] [35].

Particularly, the three previously mentioned models for delay, wire-length and power estimation were appropriately extended to be aware of 3-D reconfigurable architectures. Towards this direction, the platform’s resource graph was annotated with the TSV connections. The procedure depicted at Fig. 15 is flexible, in that it can estimate the power consumption for a wide variety of 3-D reconfigurable architectures; hence throughout this article it was also applied to the rest (reference) tool flows [12] [13]. Although more accurate techniques for power estimation can be found in relevant publications, this is beyond the scopes of our work because these techniques are either not general-purpose (i.e. applicable to different 3-D architectures), or they impose mentionable computational complexity. On contrast, the proposed activity-based approach is fast enough, as it does not require extensive simulations.

V. PERFORMANCE EXPLORATION AND COMPARISON RESULTS

This section quantifies the efficiency of introduced framework underlying compared to the state-of-the-art relevant algorithms. The underline 3-D FPGA follows the well-established hierarchical island-style architecture discussed at Section II. Two flavors of our framework, with and without fast placement, mentioned as “fast” and “normal”, respectively, are evaluated against to two publicly available flows, where netlist partitioning is computed with simulated annealing [12] and hMetis [13] algorithms, respectively. Similar to our framework, the reference flows rely on simulated annealing placer and pathfinder router. For comparison purposes, we quantify also the efficiency of the “normal” flavor without the algorithmic customization feature (referred as “normal without ANN”). In such a case, the value of each of the algorithmic parameters presented in Table III is set equal to the average value of its respective range of possible values (depicted at the last column of Table III). Unfortunately, we cannot provide comparisons against to the rest flows summarized at Table I (MEVA-3D [11], VPR3D [14] and 3D-Tree [15]), as these tools are not publicly available.

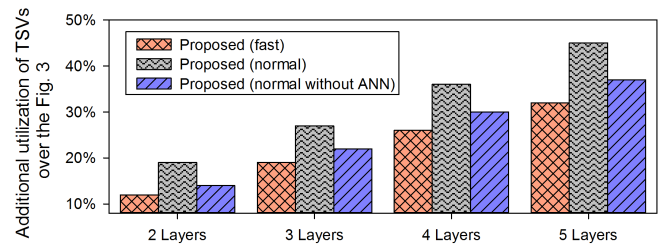


Fig. 16: Percentage of additionally utilized TSVs in comparison with results summarized at Fig. 3.

The evaluation is performed with the 20 largest MCNC benchmark circuits [29]. In order to further increase the complexity of these benchmarks, a special pre-processing step is applied by appropriately combining them in groups of 10 with a technique discussed in [39]. This results to an average complexity of 5,910 CLBs per benchmark⁶. Although more complex benchmark suites are available (e.g. TITAN [40]), these benchmarks are not compatible to existing TPR-based flows [12] [13], as they assume devices with heterogeneous blocks (e.g. memories, DSPs, CPUs, etc). Table IV summarizes the array size per benchmark when different 3-D FPGA devices are considered. These sizes refer to the minimum array required for successful P&R with existing tools [12] [13]⁷, whereas for sake of completeness the target platform per benchmark is identical among flows. Additionally, whenever it is not mentioned explicitly, the results discussed at this section are average values among the 20 MCNC benchmarks, while for demonstration purposes these results are plotted in normalized manner over the corresponding design metric for the 2-D FPGA. The results for the 2-D FPGA are computed based on a widely accepted software in academia, named VPR [20], which also relies on simulated annealing placement and negotiated pathfinder routing algorithms. Finally, in order to avoid the impact of CAD noise [41], all the algorithms are tuned with constant seed values.

A. Analysis of utilized TSVs

The efficiency of introduced framework relies on manipulating in a better way than existing tools the fabricated TSVs. In order to study more thoroughly this parameter, Fig. 16 plots the additional utilization of TSVs for the alternative flavors of the proposed framework, in comparison to the results discussed at Fig. 3. This analysis exhibits a predictable but reasonable picture. Specifically, the utilization of TSVs increases monotonically with the number of device layers compared to the state-of-the-art TPR tool [13]. This occurs mainly because the refinement loops discussed at Fig. 4 enable our framework to estimate more accurately the actual demand for inter-layer connections during the partitioning and placement algorithms. Thus, it is possible to tune appropriately the algorithm’s aggressiveness to form more or fewer connections between CLBs assigned to different layers. Precisely, the introduced framework (“normal” flavor) utilizes on average 19%–45% additional TSVs in comparison to the reference flows, which

⁶On average each benchmark consists of 29,496 4-input LUTs.

⁷We achieve average utilization of logic resources ranging between 85%–90%.

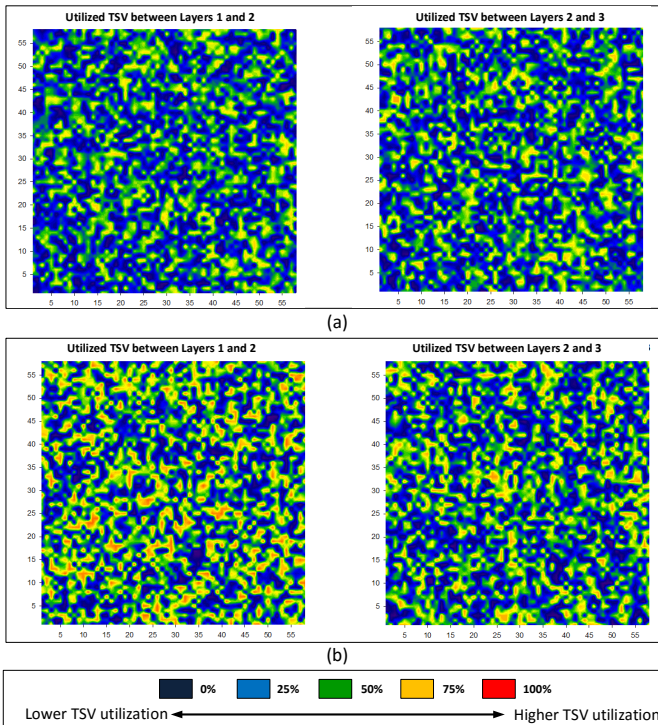


Fig. 17: Spatial distribution of utilized TSVs for ex1010×10 benchmark with: (a) the TPR tool [13] and (b) the proposed framework.

in turn enables the employed CAD algorithms to form shorter routing paths.

Apart from the total number of utilized TSVs, their spatial distribution over each layer is also important for addressing the routing congestion problem. Fig. 17 gives the variation of this parameter regarding the ex1010×10 benchmark mapped onto an FPGA with three layers. For this analysis, both the existing TPR [13] (Fig. 17(a)) and the “normal” flavor of our framework (Fig. 17(b)), are employed.

Each point (x_i, y_i) of this contour graph depicts the percentage⁸ of utilized TSVs inside a 3-D SB placed on spatial location (x_i, y_i) . This graph indicates that the utilization of TSVs varies between two arbitrary points (x_1, y_1, z_1) and (x_2, y_2, z_2) of the device, even for SBs placed on adjacent locations within the same layer. Such a conclusion, in conjunction to the hardware uniformity of FPGAs, leads to a mentionable wastage of fabricated TSVs, especially for the existing flows that rely on a min-cut netlist partitioning. More specifically, only a small portion (about 5% on average) of the SBs utilize all their TSVs (i.e. utilize W_V TSVs per SB) when existing algorithms are employed to perform physical implementation. On the other hand, the introduced framework achieves to increase this percentage by 3–4× in comparison to the results depicted at Fig. 17(a). Note that our framework does not lead to unroutable designs, since the majority of SBs utilize on average 50%-75% of the fabricated TSVs.

⁸As we have mentioned at Section II, there are 3 TSVs per SB ($W_V = 3$).

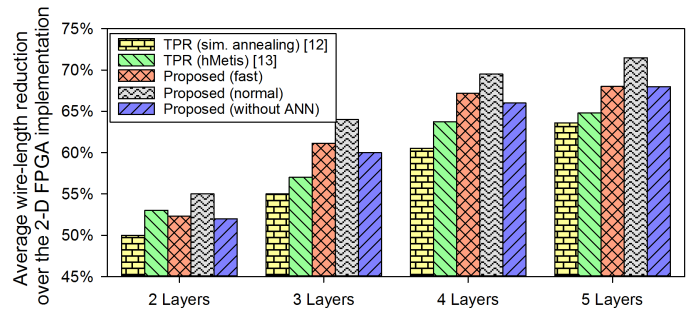


Fig. 18: Average wire-length for successful routing.

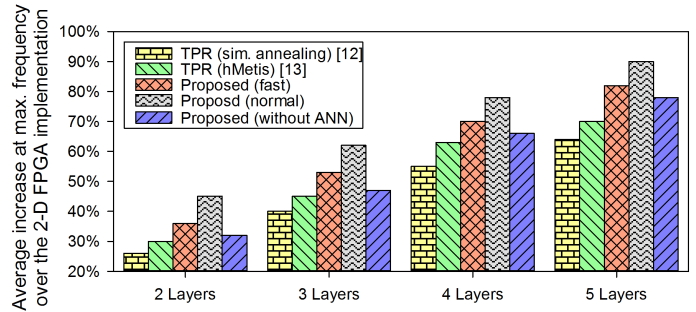


Fig. 19: Average maximum operation frequency among frameworks.

B. Evaluation of Performance Enhancement

The primary advantage of using a 3-D architecture is the spatial locality (i.e. reduced Manhattan distance) among resources assigned to different layers, which in turn is expected to improve application’s critical path delay and power consumption. In order to study more thoroughly this topic, Fig. 18 plots the total wire-length for successful P&R. Based on this analysis we might conclude that the introduced framework outperforms existing TPR-based flows [12] [13], as we achieve an additional reduction at total wire-length on average 9%–14%. These gains are due to the better manipulation of TSVs during the netlist partitioning and routing algorithms. Furthermore, we have to highlight that the algorithms of our framework surpass existing tools, even without customization. Specifically, based on Fig. 18, the flavor without ANN achieves average wire-length reduction by 7% and 3%, as compared to the “TPR (sim. annealing) [12]” and “TPR (hMetis) [13]”, respectively.

The shorter routing paths impose lower resistance (R) and capacitance (C) values, which in turn lead to smaller critical path delay. To study this metric, Fig. 19 gives the average variation of maximum operation frequency for different 3-D FPGAs. The timing analysis both for the 2-D and 3-D devices is performed with Elmore delay model [38], in view of its high fidelity with respect to the actual circuit delay [42].

The results of this analysis indicate that the introduced framework achieves higher operation frequency than the TPR-based flows [12] [13]. More specifically, the “normal” flavor achieves an average increase to the clock’s speed by 35% and 53%, as compared to the “TPR (sim. annealing) [12]” and “TPR (hMetis) [13]” flows, respectively. These gains are inline with those already discussed about the wire-length reduction due to the better manipulation of fabricated TSVs.

The wire-length reduction leads also to power saving, as

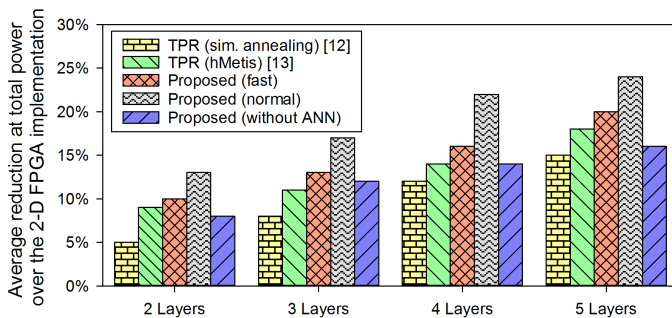


Fig. 20: Average power consumption assuming maximum clock frequency.

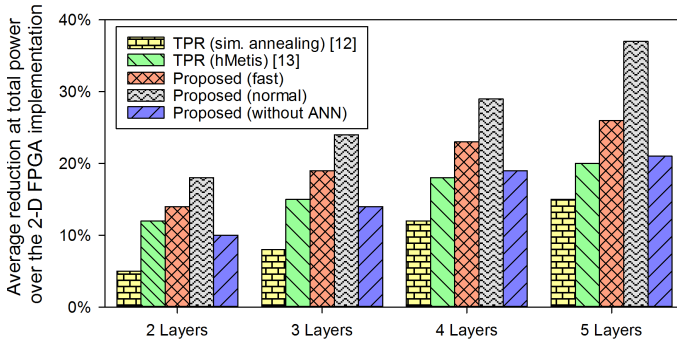


Fig. 21: Average power consumption assuming identical clock frequency among frameworks.

it is depicted at Fig. 20. For this analysis, each of the alternative flows is evaluated with the procedure proposed at Fig. 15. Based on this graph we conclude that our framework outperforms existing flows. Specifically, the “normal” flavor achieves additional power savings in comparison with “TPR (hMetis) [13]” by 47% on average, while the corresponding power reduction as compared to the “TPR (sim. annealing) [12]” is almost $2\times$.

Moreover, it is well-worth to mention that the gains in terms of maximum operation frequency and power consumption are higher, when the proposed framework incorporates the feature of algorithmic customization through the ANN. Precisely, in such a case, the average improvement for the critical path delay and the total power consumption are 26% and 53%, respectively, compared to the case where the ANN is not employed. Such a conclusion highlights the necessary to perform algorithmic customizations, similar to those discussed throughout this article.

Note that the power savings discussed previously are complementary to the gains in application’s maximum operation frequency. Hence, one might expect that the proposed framework improves further either the critical path delay under the same power budget, or the power savings for identical operation frequency. This topic is studied at Fig. 21, where the average power consumption per benchmark is computed assuming the minimum clock frequency among frameworks. From this analysis we might conclude that our framework achieves to reduce further the average power consumption as compared to the state-of-the-art “TPR (hMetis) [13]” by 64%.

Apart from the performance metrics discussed so far, the execution run-time of CAD algorithms is also of high im-

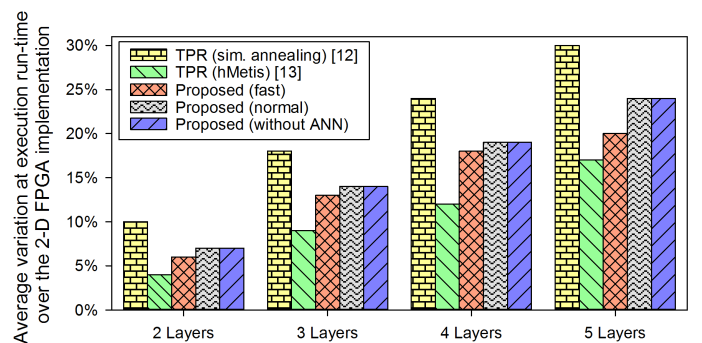


Fig. 22: Execution run-time for performing physical implementation onto 3-D FPGAs.

portance. In order to study more thoroughly this aspect, Fig. 22 plots the execution run-time for performing physical implementation regarding the alternative flows. These results indicate that the fastest tool flow is the “TPR (hMetis) [13]”, which requires almost the half time as compared to the existing “TPR (sim. annealing) [12]” tool. Regarding the “normal” flavor, it imposes almost an $1.6\times$ overhead at execution run-time versus the fastest tool flow. This penalty occurs mainly due to the additional computational complexity imposed by the employed tabu partitioning algorithm, as compared to the hMetis-based approach found in [13]. One more conclusion might be derived from this analysis. Specifically, there is a monotonic increase at the execution run-time as we employ 3-D architectures with additional layers. Although the additional layers result to smaller array sizes per layer, as it is depicted at Table IV, the number of TSVs increases the complexity for netlist routing. Finally, it is well-worth to highlight that there is no penalty for the algorithmic customization, since the execution of ANN is very fast after it has been trained.

VI. CONCLUSIONS

A novel framework for application implementation onto 3-D FPGAs, was introduced. This framework incorporates mechanisms to customize the employed CAD algorithms in order to better manipulate constraints posed either by the target application, or the underline 3-D architecture. Experimental results with various benchmarks prove the effectiveness of the proposed solution, compared to existing state-of-the-art flows in academia. Specifically, we achieve average gains in terms of total wire-length, critical path delay and power consumption by 9%, 35% and 47%, respectively.

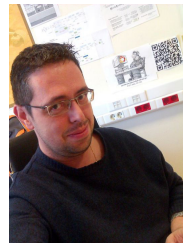
ACKNOWLEDGMENT

This work is partially supported by “TEACHER: TEach AdvanCED Reconfigurable architectures and tools” project funded by DAAD (2014). Also, this work is partially supported by the E.C. funded program AEGLE under H2020 Grant Agreement No: 644906, <http://www.aegle-uhealth.eu>.

REFERENCES

- [1] ITRS, “International technology road map for semiconductors-interconnect tsv roadmap,” 2013. [Online]. Available: <http://www.itrs.net>
- [2] N. Magen, A. Kolodny, U. Weiser, and N. Shamir, “Interconnect-power dissipation in a microprocessor,” in *Proc. of the Int. Workshop on System Level Interconnect Prediction*. USA: ACM, 2004, pp. 7–13.

- [3] A. Papanikolaou, D. Soudris, and R. Radojicic, Eds., *Three Dimensional System Integration: IC Stacking Process and Design*. Springer, 2011.
- [4] V. F. Pavlidis and E. G. Friedman, *Three-dimensional Integrated Circuit Design*. CA, USA: Morgan Kaufmann Publishers Inc., 2009.
- [5] K. Leijten-Nowak and J. L. van Meerbergen, "An fpga architecture with enhanced datapath functionality," in *Proc. of the Int. Symp. on Field Programmable Gate Arrays*. USA: ACM, 2003, pp. 195–204.
- [6] Tezzaron, 2014. [Online]. Available: http://www.tachyonsemi.com/about/PhotoAlbum/Products/3D_FPGA.html
- [7] Xilinx, 2014. [Online]. Available: <http://www.xilinx.com/>
- [8] M. Santarini, "Xilinx 16nm ultrascale+ devices yield 2-5x performance/watt advantage," in *XCell Journal*, vol. 90, 1st quarter 2015, pp. 9–15. [Online]. Available: <http://www.xilinx.com/publications/archives/xcell/Xcell90.pdf>
- [9] M. Lin, A. El Gamal, Y.-C. Lu, and S. Wong, "Performance benefits of monolithically stacked 3-d fpgas," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Trans. on*, vol. 26, no. 2, pp. 216–229, Feb 2007.
- [10] K. Siozios, V. F. Pavlidis, and D. Soudris, "A novel framework for exploring 3-d fpgas with heterogeneous interconnect fabric," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 5, no. 1, pp. 4:1–4:23, Mar. 2012.
- [11] MEVA-3D, 2014. [Online]. Available: http://cadlab.cs.ucla.edu/three_d/3dic.html
- [12] C. Ababei, Y. Feng, B. Goplen, H. Mogal, T. Zhang, K. Bazargan, and S. Sapatnekar, "Placement and routing in 3d integrated circuits," *Design Test of Computers, IEEE*, vol. 22, no. 6, pp. 520–531, Nov 2005.
- [13] C. Ababei, H. Mogal, and K. Bazargan, "Three-dimensional place and route for fpgas," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Trans. on*, vol. 25, no. 6, pp. 1132–1140, June 2006.
- [14] A. Hahn Pereira and V. Betz, "Cad and routing architecture for interposer-based multi-fpga systems," in *Proc. of the Int. Symp. on Field-programmable Gate Arrays*. USA: ACM, 2014, pp. 75–84.
- [15] V. Pangracious, E. Amouri, Z. Marakchi, and H. Mehrez, "Architecture level optimization of 3-dimensional tree-based FPGA," *Microelectronics Journal*, vol. 45, no. 4, pp. 355–366, 2014.
- [16] K. Siozios and D. Soudris, "A tabu-based partitioning and layer assignment algorithm for 3-d fpgas," *IEEE Embed. Syst. Lett.*, vol. 3, no. 3, pp. 97–100, Sep. 2011.
- [17] —, "A power-aware placement and routing algorithm targeting 3d fpgas," *J. Low Power Electronics*, vol. 4, no. 3, pp. 275–289, 2008.
- [18] N. Selvakkumaran and G. Karypis, "Multiobjective hypergraph-partitioning algorithms for cut and maximum subdomain-degree minimization," *Trans. Comp.-Aided Des. Integr. Cir. Sys.*, vol. 25, no. 3, pp. 504–517, Nov. 2006.
- [19] S. Vassiliadis and D. Soudris, *Fine- and Coarse-Grain Reconfigurable Computing*, 1st ed. Springer Publishing Company, Incorporated, 2007.
- [20] V. Betz, J. Rose, and A. Marquardt, Eds., *Architecture and CAD for Deep-Submicron FPGAs*. USA: Kluwer Academic Publishers, 1999.
- [21] S. Gupta, M. Hilbert, S. Hong, and R. Patti, "Techniques for Producing 3D ICs with High-Density Interconnect," in *VLSI Multi-Level Interconnection Conference*, 2004.
- [22] 2014. [Online]. Available: http://www.eecg.utoronto.ca/vpr/architectures/architecture_table.html
- [23] W. Zhao and Y. Cao, "New generation of predictive technology model for sub-45nm design exploration," in *Proc. of the 7th Int. Symposium on Quality Electronic Design*, USA, 2006, pp. 585–590.
- [24] D. H. Kim and S. K. Lim, "Through-silicon-via-aware delay and power prediction model for buffered interconnects in 3d ics," in *Int. Workshop on System Level Interconnect Prediction*, USA, 2010, pp. 25–32.
- [25] K.-L. Du and M. N. S. Swamy, Eds., *Neural Networks and Statistical Learning*. Springer, 2014.
- [26] R. D. Reed and R. J. Marks, *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. USA: MIT Press, 1998.
- [27] "Benchmark Designs For The Quartus University Interface Program (QUIP)," Jun. 2005. [Online]. Available: https://www.altera.com/support/software/download/altera_design/quip/quip-download.jsp
- [28] H. Demuth, M. Beale, and M. Hagan, "Neural network toolbox™ 6 user's guide," 2009.
- [29] S. Yang, "Logic synthesis and optimization benchmarks guide," Technical Report 1991-IWLS-UG-Saeyang, 1991.
- [30] D. Mitra, F. Romeo, and A. Sangiovanni-Vincentelli, "Convergence and finite-time behavior of simulated annealing," in *Decision and Control, 1985 24th IEEE Conference on*, Dec 1985, pp. 761–767.
- [31] C. liang Eric Cheng, "Risa: Accurate and efficient placement routability modeling," in *Computer-Aided Design, 1994., IEEE/ACM International Conference on*, Nov 1994, pp. 690–695.
- [32] M. Feuer, "Connectivity of random logic," *Computers, IEEE Transactions on*, vol. C-31, no. 1, pp. 29–33, Jan 1982.
- [33] A. Gamal, "Two-dimensional stochastic model for interconnections in master slice integrated circuits," *Circuits and Systems, IEEE Transactions on*, vol. 28, no. 2, pp. 127–138, Feb 1981.
- [34] J. Lamoureux and S. J. E. Wilton, "Activity estimation for field-programmable gate arrays," in *Int. Conf. on Field Programmable Logic and Applications*, 2006, pp. 1–8.
- [35] K. K. W. Poon, S. J. E. Wilton, and A. Yan, "A detailed power model for field-programmable gate arrays," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 10, no. 2, pp. 279–302, Apr. 2005.
- [36] H. Sidiropoulos, K. Siozios, P. Figuli, D. Soudris, M. Hübner, and J. Becker, "Jitpr: A framework for supporting fast application's implementation onto fpgas," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 6, no. 2, pp. 7:1–7:12, Aug. 2013.
- [37] L. McMurchie and C. Ebeling, "Pathfinder: A negotiation-based performance-driven router for fpgas," in *Proceedings of the 1995 ACM Third International Symposium on Field-programmable Gate Arrays*. NY, USA: ACM, 1995, pp. 111–117.
- [38] D. Hodges, H. Jackson, and R. Saleh, *Analysis and Design of Digital Integrated Circuits*, 3rd ed. NY, USA: McGraw-Hill, Inc., 2004.
- [39] J. Lamoureux, "Benchmark circuits," 2014. [Online]. Available: <http://www.ece.ubc.ca/julienl/scripts/stitch-blifs.pl>
- [40] K. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, "Titan: Enabling large and complex benchmarks in academic cad," in *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, Sept 2013, pp. 1–8.
- [41] W. Shum and J. H. Anderson, "Analyzing and predicting the impact of cad algorithm noise on fpga speed performance and power," in *Proc. of the Int. Symp. on Field Programmable Gate Arrays*. USA: ACM, 2012, pp. 107–110.
- [42] K. Boese, A. Kahng, B. McCoy, and G. Robins, "Fidelity and near-optimality of elmore-based routing constructions," in *Computer Design: VLSI in Computers and Processors, Int. Conf. on*, Oct 1993, pp. 81–84.



Kostas Siozios received his Diploma, Master and Ph.D. Degree in Electrical and Computer Engineering from the Democritus University of Thrace, Greece, in 2001, 2003 and 2009, respectively. Currently he is working as research associate in the National Technical University of Athens, Greece. His research interests include CAD algorithms, reconfigurable architectures, 3-D integration technology, network-on-chip and parallel architectures. He has published more than 100 papers in international journals and conferences. Also, he has coauthor/coeditor in 5 books of Kluwer and Springer. The last years he works as principal investigator in numerous research projects funded from the European Commission (EC), European Space Agency (ESA), as well as the Greek Government and Industry.



Dimitrios Soudris received the Diploma and PhD degree in electrical & computer engineering from the University of Patras, Greece, in 1987 and 1992, respectively. Since 1995 and for 13 years, he has served as a professor in the Department of Electrical and Computer Engineering, Democritus University of Thrace, Greece. He is currently an associate professor with the School of Electrical and Computer Engineering, National Technical University of Athens, Greece. His research focuses on embedded systems design, low power VLSI design, and reconfigurable architectures. He has published more than 380 papers and is the coauthor/coeditor of six Kluwer/Springer books. He is a leader and principal investigator in numerous research projects funded from the Greek Government and Industry, European Commission and European Space Agency. He is a member of the VLSI Systems and Applications Technical Committee of IEEE CAS. He is a member of the IEEE and ACM.