A Learning Bridge from Architectural Synthesis to Physical Design for Exploring Power Efficient High-Performance Adders

Subhendu Roy[‡], Yuzhe Ma[†], Jin Miao[‡], and Bei Yu[†]

[‡]Cadence Design Systems, San Jose, CA, USA
[†]CSE Department, The Chinese University of Hong Kong, NT, Hong Kong
{subhroy,jmiao}@cadence.com, {yzma,byu}@cse.cuhk.edu.hk

***Abstract*—In spite of maturity to the modern electronic design automation (EDA) tools, optimized designs at architectural stage may become sub-optimal after going through physical design flow. Adder design has been such a long studied fundamental problem in VLSI industry yet designers cannot achieve optimal solutions by running EDA tools on the set of available prefix adder architectures. In this paper, we enhance a state-of-the-art prefix adder synthesis algorithm to obtain a much wider solution space in architectural domain. On top of that, a machine learning based design space exploration methodology is applied to predict the Pareto frontier of the adders in physical domain, which is infeasible by exhaustively running EDA tools for innumerable architectural solutions. Experimental results demonstrate that our framework can achieve near-optimal delay vs. power/area Pareto frontier over a wide design space, bridging the gap between architectural and physical designs.**

I. Introduction

In the last decade, the industrial EDA tools have advanced towards optimality, especially at the individual stages of VLSI design cycle. Nevertheless, with growing design complexity and aggressive technology scaling, physical design issues have become more and more complex. As a result, the constraints and the objectives of higher layers, such as the system or logic level, are very difficult to map into those of lower layers, such as physical design, and vice-versa, thereby creating a *gap* between the optimality at the logic stage and the physical design stage. This necessitates the innovation of data-driven methodologies, such as machine learning [1]–[5], to bridge this gap.

Adder design is one of the fundamental problems in digital semiconductor industry, and its main bottleneck (in terms of both delay and area) is the carry-propagation unit. This unit can be realized by hundreds of thousands of parallel prefix structures, but hard to evaluate the final metrics without running through physical design tools. Historically, regular adders [6]–[9] have been proposed for achieving the corner points in terms of various metrics in architectural stage. The main motivation for structural regularity was the ease of manual layout, but EDA tools now taking care of all physical design aspects, the regularity is no longer essential. Moreover, the extreme corners do not map well to the physical design metrics after synthesis, placement and routing. To address this gap between prefix adder synthesis and actual physical design of the adders, custom adders are typically designed by tuning parameters, such as gate-sizing, buffering etc., targeting at the optimization of power/performance metrics for a specific technology library [10], [11]. However, this custom approach (i) needs significant engineering effort, (ii) is not flexible to Engineering Change Order (ECO), and (iii) does not guarantee the optimality.

The algorithmic synthesis approach resolves the first two issues of the custom approach, by adding more flexibility to the late ECO changes and reducing the engineering effort. Based on the number of solutions, the existing adder synthesis algorithms can be broadly classified into two categories. The first and the most common approach is to generate a single prefix network for a set of structural constraints, such as the logic level, fan-out etc. Several algorithms have been proposed to minimize the size of the prefix graph (s) under given bit-width (n) and logic-level (L) constraints [12]–[15]. Closed form theoretical bounds for size-optimality are provided by [16] for $L \geq 2log_2 n - 2$. [17] has given more general bound for prefix graph size, but when L is reduced to $log_2 n$, a pre-requisite for high-performance adders, there is no closed form bound for s. [18] presents a polynomial-time algorithm for generating prefix graph structures by restricting both logic-level and fan-out. The limitations in these approaches are two-fold, (i) this restricted set of structures is not capable of exploring the large solution space, and (ii) since it is very hard to analytically model the physical design complexities, such as wire-length and congestion issues, the physical design metrics, such as the area, power, delay etc., may not be mapped well to the prefix structure metrics, such as the size, max-fan-out (mfo) etc. This motivates the second category of algorithms where thousands of prefix adder solutions can be generated and explored for synthesis and physical design in the commercial EDA tools.

One such approach is [19] which presents an exhaustive bottom-up enumeration technique with several pruning strategies to generate innumerable prefix structure solutions. However, it has two issues, (i) this approach cannot provide solutions in several cases for restricted fan-out, which can control the congestion and load-distribution during physical design [18]. As a result, it may still miss the *good* solution space to a large extent, and (ii) it is computationally very intensive to run all solutions through synthesis, placement and routing.

In this paper, we enhance the algorithm in [19] to generate adders under any arbitrary mfo constraint, which enables a *wider* adder solution space in *logical* form. To tackle the high computational effort during the physical design flow, we propose to use machine learning to predict the power, area and delay of adders in *physical* solution space. Features being used include prefix structure attributes and the EDA tool settings. We employ a *quasi-random* sampling approach to select representative prefix adders out of the hundreds of thousands of prefix structures. Furthermore, we develop a Pareto frontier driven machine learning methodology to achieve rich adder solutions with trade-offs among power, area, and delay. To the best of our knowledge, this is the first attempt to bridge architectural synthesis with physical design in the field of EDA using machine learning. Our main contributions are summarized as follows:

- We develop the first comprehensive framework for optimal adder search by machine learning methodology bridging the prefix architecture synthesis to the final physical design.
- An enhancement to a state-of-the-art prefix adder algorithm [19] is proposed to optimize the prefix graph size for restricted fan-out and explore a wider solution space.
- We develop machine learning models for prefix adders, guided by quasi-random data sampling with features considering architectural

978-1-5090-6023-8/17/$31.00

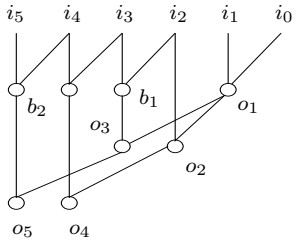


Fig. 1: 6 bit prefix adder network.

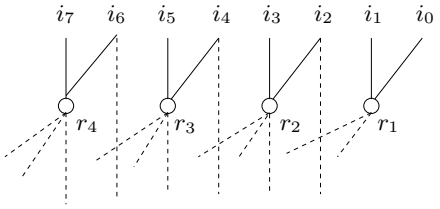


Fig. 2: Imposing semi-regularity.

attributes and EDA tool settings.

- A design space exploration method is proposed to generate the Pareto frontier for delay vs. power/area over a wide design space.

The rest of the paper is organized as follows. Section II presents the background of prefix adder synthesis followed by our prefix graph generation algorithm. Machine learning driven design space exploration for high-performance adders is described in Section III. Section IV presents the experimental results followed by conclusion in Section V.

II. Prefix Adder Synthesis

In this section, we first provide the background of the prefix adder synthesis problem. Then we present a brief discussion on the algorithm presented in [19], which we enhance to our **Prefix Graph Generation (PGG)** algorithm to synthesize the prefix adder network.

A. Preliminaries

An n bit adder accepts two n bit addends $A = a_{n-1}..a_1a_0$ and $B = b_{n-1}..b_1b_0$ as input, and computes the output sum $S = s_{n-1}..s_1s_0$ and carry out $C_{out} = c_{n-1}$, where $s_i = a_i \oplus b_i \oplus c_{i-1}$ and $c_i = a_ib_i + a_ic_{i-1} + b_ic_{i-1}$. The simplest realization for the adder network is the ripple-carry-adder, but with logic level $n-1$, which is too slow. For faster implementation, carry-lookahead principle is used to compute the carry bits. Mathematically, this can be represented with bitwise (group) generate function g (G) and propagate function p (P) by the Weinberger's recurrence equations as follows [20]:

- Pre-processing (inputs): Bitwise generation of g, p

$$g_i = a_i \cdot b_i \text{ and } p_i = a_i \oplus b_i. \tag{1}$$

- Prefix processing: This part is the main carry-propagation component where the concept of generate/propagate is extended to multiple bits and $G_{[i:j]}$, $P_{[i:j]}$ ($i \geq j$) are defined as

$$P_{[i:j]} = \begin{cases} p_i, & \text{if } i = j, \\ P_{[i:k]} \cdot P_{[k-1:j]}, & \text{otherwise}, \end{cases} \tag{2}$$

$$G_{[i:j]} = \begin{cases} g_i, & \text{if } i = j, \\ G_{[i:k]} + P_{[i:k]} \cdot G_{[k-1:j]}, & \text{otherwise.} \end{cases} \tag{3}$$

The associative operation o is defined for (G, P) as:

$$\begin{aligned}(G,P)_{[i:j]} &= (G,P)_{[i:k]} \; o \; (G,P)_{[k-1:j]} \\ &= (G_{[i:k]} + P_{[i:k]} \cdot G_{[k-1:j]}, P_{[i:k]} \cdot P_{[k-1:j]}).\end{aligned} \tag{4}$$

- Post-processing (outputs): Sum/Carry-out generation

$$s_i = p_i \oplus c_{i-1}, \quad c_i = G_{[i:0]}, \text{ and } C_{out} = c_{n-1}. \tag{5}$$

The 'Prefix processing' or carry propagation network can be mapped to a prefix graph problem with inputs $i_k = (p_k, g_k)$ and outputs $o_k = c_k$, such that o_k depends on all previous inputs i_j ($j \leq k$). Any node except the input nodes is called a *prefix* node. Size of the prefix graph is defined as the number of prefix nodes in the graph. Fig. 1 shows an example of such prefix graph of 6 bit and we can see that C_{out} = c_5 = o_5 is given by

$$o_5 = (i_5 \; o \; i_4) \; o \; ((i_3 \; o \; i_2) \; o \; (i_1 \; o \; i_0)). \tag{6}$$

Size (s), logic level (L) and maximum-fan-out (mfo) for this network are respectively 8, 3 and 2. Note that here the number of fan-ins for each of the associative operation o is two, thus this is called radix-2 implementation of the prefix graph. However, there exist other options such as radix-3 or radix-4, but the complexity is very high and not beneficial in static CMOS circuits [21]. In this work, the logic levels for all output bits are $log_2 n$, *i.e.*, the minimum possible, to target high performance adders.

B. Discussion on [19]

Our PGG algorithm to generate the prefix graph structures for physical solution space exploration is based on [19]. So it is imperative to first discuss about [19]. However, we omit the details and only mention the key points of [19] due to space constraint.

[19] is an exhaustive bottom-up and pruning based enumeration technique for prefix adder synthesis. This work presented an algorithm to generate all possible $n+1$ bit prefix graph structures from any n bit prefix graph. Then this algorithm is employed in a bottom-up fashion (from 1 bit adder to 2 bit adders, then from all 2 bit adders to 3 bit adders, and so on) to synthesize prefix graphs of any bit-width. As a result, scalability issue arises due to the exhaustive nature of the algorithm, which is then tackled by adopting various pruning strategies to scale the approach. However, the pruning strategies are not sufficient to scale the algorithm well for different fan-out constraints. So when it intends to find the solutions for higher bit adders, the intermediate adder solutions that need to be generated are often huge. Consequently, it fails to get fan-out restricted (e.g. when $mfo = 8, 10, 12$ etc. for 64 bit adders) solutions even with 72GB RAM due to the generation of innumerable intermediate solutions [18]. Pruning strategies, such as size-bucketing [19], helps to achieve solutions in some cases, but with sub-optimality. So design space-exploration based on this algorithm can miss a significant spectrum of the adder solutions.

C. Our PGG Algorithm

To better explore the wide design space of adders, in this paper we have enhanced [19] for different fan-out constraints by incorporating more pruning techniques.

1) Semi-Regularity in Prefix Graph Structure

The first strategy is to enforce a sort of regularity in the prefix graphs. For instance, regular adders, such as Sklansky, Brent-Kung, have the inherent property that the consecutive input nodes (even and odd) are combined to create the prefix nodes at the logic level 1. In our approach, we constrain this regularity for those prefix nodes (logic level 1). To explain this, let us consider Fig. 2. We can see that prefix nodes r_1, r_2, r_3 and r_4 are constructed by consecutive even-odd nodes. For instance, i_0 and i_1 are used to construct r_1. But with this structural constraint, we are not allowed to construct any node by combining i_1 and i_2 as done in Kogge-Stone adders. Note that the sub-structure, as shown in Fig. 2, is a part of some regular adders like Sklansky adder, and is imposed in our prefix structure enumeration. However, unlike Sklansky adders, we allow irregularity at all logic levels except $level = 1$.

By doing so, we have observed that (i) the solution quality (or size of the prefix graph under same L and mfo) is not degraded in comparison to [19], but (ii) it is able to reduce the search space significantly. However, imposing similar regularity for $level > 1$ lead to sub-optimal solutions for higher bit adders, such as $n \geq 16$.

TABLE I: Comparison with [19] for 64 bit adders

mfo	Our Approach		Approach in [19]	
	size	Run-time (s)	size	Run-time (s)
4	244	302	252	241
6	233	264	238	212
8	222	423	-	-
12	201	193	-	-
16	191	73	192	149
32	185	0.04	185	0.04

2) Level Restriction in Non-trivial Fan-in

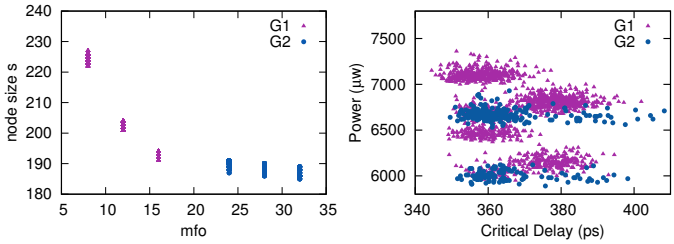
Each of the prefix node N (a:b), where a is the most-significant bit (MSB) and b is the least-significant-bit (LSB), is constructed by connecting the trivial fan-in N_{tr} (a:c) having same MSB as N, and the non-trivial fan-in N_{non-tr} ($c-1$:b). For instance, in Fig. 1, o_3 and b_2 are respectively the non-trivial fan-in and the trivial fan-in node for the prefix node o_5. In the bottom-up enumeration technique, we put another additional restriction that the level of the trivial fan-in node is always less or equal to that of the non-trivial fan-in node, *i.e.*, $level(N_{tr}) \leq level(N_{non-tr})$. Note that this sort of structural restriction is also inherent in regular adders, such as Sklansky or Brent-Kung adders.

In a nutshell, our PGG algorithm is a blend of regular adders and [19]. We borrow some properties of regular adders to enforce in [19] for reducing its huge search space without hampering the solution quality. To illustrate this, we have obtained the binary for [19] from the authors, and first compared our result for lower bit adders, such as $n = 16, 32$. We got the solutions with same minimum size, which *proves that our structural constraints have not degraded the solution quality*. However, for higher bit adders, we get better solution quality than [19] as shown in TABLE I.

Column 1 presents the mfo constraint, while columns 2 and 3 respectively show the size and run-time for our enhanced algorithm, and the corresponding entries for [19] are respectively represented in columns 4 and 5. In general, when fan-out is relaxed or mfo is higher, the run-time is less due to relaxed size-pruning as explained in [19]. Note that [19] cannot generate solutions for $mfo = 8, 12$ due to generation of innumerable intermediate solutions as explained in [18]. On the contrary, our structural constraints can do a pre-filtering of the potentially futile solutions, thereby allowing relaxed size-pruning and size-bucketing to search for more effective solution space. In terms of run-time, it is slightly worse in a few cases, but importantly, this generation is a one-time process, and this run-time is negligible in comparison to the design space exploration by the physical design tools. So our imposed structural restrictions (i) do not degrade the solution quality, (ii) achieve better solution sizes for all mfo than [19] for higher bit adders which could not even generate solutions in all cases, and (iii) help to obtain wider physical solution space to be demonstrated in Section IV-A.

III. Bridging Architectural Solution Space to Physical Solution Space

In this section, we first discuss the gap between the prefix architectural solution space and physical design solution space in adders, which motivates the need of the machine learning based approach for optimal adder exploration. We then present the quasi-random data sampling for our learning models. Next a domain-knowledge based feature selection details are presented. Finally, we propose our learning based Pareto frontier exploration methodology for prefix adder architectures, with trade-off between delay vs. power/area, to tackle the intractable runtime that would have been resulted by exhaustively running physical design flow for all individual prefix network solutions.



(a) (b)

Fig. 3: Gap between prefix structure and physical design of adders: (a) Architectural solution space; (b) Physical solution space.

A. Gap Between Logic and Physical Design

Since we focus on high-performance adders and explore the prefix adders of logic level $L = log_2 n$, the metrics at this architecture stage are prefix node size s and max fan-out mfo. These two metrics are conflicting, *i.e.*, if we reduce mfo, s increases and vice-versa. Similar competing relationship exists between delay and power/area after physical design. It should be stressed that power and s are correlated, and mfo indirectly controls the timing as more restricted fan-out can mitigate congestion and load-distribution, thereby improving the delay of the adder. However, this relationship between architectural synthesis and physical design is approximate, and not a very high-fidelity one.

To demonstrate this, we plot node size s vs. mfo and power vs. delay in Fig. 3 for several 64 bit adder solutions. For this experiment, we have generated the prefix architecture solutions by PGG, and the final power/delay numbers are obtained by running those solutions through EDA tools as explained later in Section IV. In Fig. 3(a), we broadly categorize the solutions into 2 groups, (i) G_1 with higher node size and lower mfo, and (ii) G_2 with lower node size and higher mfo. In Fig. 3(b), the solutions are projected into the physical solution space, restoring the group information. The key observations here are firstly, *there is a correlation between architectural solution space and physical design solution space*. For instance, the solutions from G_1 are mostly on the upper side, and those of G_2 are mostly on the lower side in Fig. 3(b), thereby indicating a correspondence between s and power. Nevertheless, *it is not completely reliable*. For example, (i) the delay numbers for G_1 and G_2 are very much spread, (ii) a cluster can be observed where the solutions from G_1 and G_2 are mixed up in Fig. 3(b), and (iii) several solutions of G_1 are better than several solutions of G_2 in power, which is not in accordance with the metrics at the prefix adder architecture stage. So we can not utterly rely on architectural solution space to achieve the optimal output in physical solution space.

However, since our algorithm generates **hundreds of thousands** of prefix graph structures, it is intractable to run synthesis and physical design flows for even a small percentage of all available prefix adder architectures. To address this **fidelity gap** between the two design stages and the high computational cost together, we come up with a novel machine-learning guided design space exploration as replacement of exhaustive search.

B. Quasi-random Sampling for Learning Model

The adversity in building the learning model is how to choose the training data as we can not afford to run the physical design flow for too many architectures, and at the same time, too few training data may degrade the model accuracy significantly. One way is to do a random sampling to pick the training data, but that may not be well representative. To tackle this, by observing a correlation between architectural and physical solution space (Section III-A), we have performed an architecture driven **quasi-random** data sampling to capture the wide design space of prefix adders as described below.

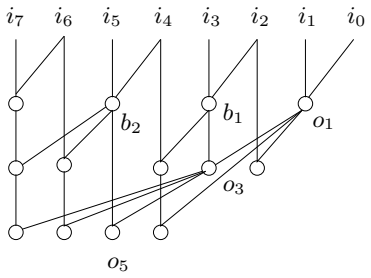


Fig. 4: Defining $spfo$ of a node.

We have mainly focused on 64 bit adders in this work as this is mostly used in today's microprocessors. From all prefix adder solutions, we sample a set of solutions for building the learning model via the quasi-random approach which is conducted by a two-level binning (mfo, s) followed by random selection, This approach aims to evenly sample the prefix adders covering different architectural bins. The primary level of binning is determined by mfo of the solutions. However, there may be thousands of architectures sharing the same mfo, so the secondary level of binning is based on s. Afterwards, adders are picked randomly from those secondary bins.

We illustrate the quasi-random sampling with the following example: given 5000 solutions with $mfo = 4$, we want to pick 50 solutions from them. Suppose these 5000 solutions have the size distribution from 244 to 258. First a random solution is picked from the bucket of the solutions ($mfo = 4$, $s = 244$). Then we pick a solution randomly from ($mfo = 4$, $s = 245$), and so on. After picking 15 solutions from each of those buckets with $mfo = 4$, we again start from the bucket ($mfo = 4$, $s = 244$). This process is repeated until we get 50 solutions. Similar procedure is done with other mfo values.

C. Feature Selection and Learning Model

We now discuss the features to be used for the learning model. Features are considered from both prefix adder structure and tool settings, with a focus on the former. We select node size and maximum-fan-out (mfo) of a prefix adder as two main features for our learning model. However, for any given mfo and node size, there will be hundreds or even thousands of different prefix architectures. Therefore, additional features are required to better distinguish individual prefix adder attributes. We define a parameter sum-path-fan-out ($spfo$) for this. Let a and b are the fan-in nodes of a node n. Then $spfo(n)$ is defined recursively as:

$$spfo(n) = \begin{cases} 0, & \text{if } n \in input, \\ sum(fo(a) + spfo(a), & \\ \quad fo(b) + spfo(b)), & \text{otherwise.} \end{cases} \quad (7)$$

Here $fo(n)$ denotes the fan-out of any node n. Consider the prefix adder structure in Fig. 4, and according to the definition we have:

$$\begin{aligned} spfo(o_1) &= sum(fo(i_0) + spfo(i_0), fo(i_1) + spfo(i_1)) \\ &= sum(1, 1) = 2, \\ spfo(b_1) &= sum(fo(i_2) + spfo(i_2), fo(i_3) + spfo(i_3)) \\ &= sum(2, 1) = 3, \\ spfo(b_2) &= sum(fo(i_4) + spfo(i_4), fo(i_5) + spfo(i_5)) \\ &= sum(2, 1) = 3. \end{aligned}$$

Therefore, we can use the recursive definition to calculate

$$\begin{aligned} spfo(o_3) &= sum(fo(o_1) + spfo(o_1), fo(b_1) + spfo(b_1)) \\ &= sum(3 + 2, 2 + 3) = 10, \\ spfo(o_5) &= sum(fo(o_3) + spfo(o_3), fo(b_2) + spfo(b_2)) \\ &= sum(3 + 10, 3 + 3) = 19. \end{aligned}$$

In our methodology, we use the $spfo$ of the output nodes which are at $\log_2 n$ level (there are 32 nodes at level 6 for 64 bit adder) as the features to characterize the prefix structures, in addition to mfo, size and target delay. The basic intuition for selecting $spfo$ of the output nodes as the features is that the critical path delay of the adder is the longest path delay from input to output. So it depends on the (i) path-lengths, which can be represented at the prefix graph stage by the logic level of the node, and (ii) the number of fan-outs driven at every node on the path. Note that we have skipped the $spfo$ of the output nodes which are not at $\log_2 n$ level as for those nodes, the path length is smaller, and those would not potentially dictate the critical path delay.

Apart from these prefix graph structural features, we also consider tool settings as other features. We have synthesized the adder structures using industry-standard EDA synthesis tool [22], where we can specify the target-delay for the adder. The tool then adopts different strategies internally to meet that target-delay which we can hardly take into account during prefix graph synthesis. Consequently, changing the target-delay can lead to different power/timing/area metrics. So we have considered target-delay as an additional feature in our learning approach.

For learning models, we explored (i) several supervised learning techniques, such as linear regression, Lasso/Ridge, Bayesian ridge model and support vector regression (SVR) with linear, polynomial and radial-basis-function (RBF) kernel, and (ii) 35 features, including 3 primary features, size, mfo and target delay (tool setting), and 32 secondary features for $spfo$. We observed that we could get an R^2 score above 0.95 for area and power even with primary features and linear models. However, we don't get good scores for delay with only primary features. Best model fitting for delay is achieved with SVR (RBF kernel) with these 3 primary and 35 secondary features. Since SVR with RBF kernel gave good MSE (mean-squared-error) scores for all metrics, delay, area and power, we have used this model throughout for design space exploration. In total 300 representative adders have been used for training/testing, and increasing the training set size further did not give us any significant improvement in model accuracy.

The model experiments gave us the following key insights: (i) **tool setting** can play an important role in building the learning models in EDA. For instance, MSE scores for area and power improve from 0.026 to 0.005, and 0.389 to 0.022 respectively when we add the 'target delay' feature in our model building, (ii) secondary features play an important role in improving the model accuracy. For instance, when we include $spfo$ features in model building, MSE score for delay improves from 0.232 to 0.164. (iii) linear models are not sufficient for modeling delay. For instance, MSE scores improve from 0.295 to 0.164 when we go from linear models to SVR with RBF kernel, with the same set of features.

D. Pareto Frontier Driven Learning

The problem of exploring the Pareto frontier of rich prefix adder space can be approached by first sampling a subset of prefix adder architectures, and generating the power, area, delay numbers of each prefix adder by running through the logic synthesis and physical design flow. Those known data set will be used as the training and testing data for supervised machine learning guided model fitting. Once the model is fitted, we can apply the exhaustive prefix adder architectures to this model and get the predicted Pareto frontier solution set. This is due to the merit of much faster runtime for a machine learning model in prediction stage than running the entire VLSI CAD flow.

However, conventional machine learning problem aims at minimizing the prediction accuracy rather than exploring a Pareto frontier out of a solution set. Improving the model accuracy does not necessarily improve the Pareto frontier and the direct use of the fitted model for Pareto frontier exploration can even miss up to 60% Pareto frontier

points [3]. We therefore need a machine learning integrated Pareto frontier exploration methodology, where the Pareto frontier selection does not rely only on the model accuracy.

In this paper, we develop a fast yet effective algorithmic methodology, enabled by regression model to explore the Pareto frontier of prefix adder solutions. We consider two spaces for Pareto frontier exploration: the delay vs. area as well as the delay vs. power. For either space, there exists a strong trade-off between the two metrics. For delay vs. power space, we propose to use a joint output Power-Delay function (PD) as the regression output rather than using any single output.

$$PD = \alpha \cdot Power + Delay. \tag{8}$$

The rationale of using scalarization [23] or the linear summation of the power and delay metrics is that such a linear relation provides a weighted bonding between the power and the delay so that by changing the α value, the regression model will try to minimize the prediction error on the more weighted axis hence leads to more accuracy on that direction. In contrast, the other metric direction will be predicted with less accuracy hence introducing some level of relaxations. It can be foreseen that changing the α value can lead to different fitting accuracies of the regression model. By sweeping α over a wide range from 0 to large positive values, each time the regression model will be fitted to predict different best solutions which altogether form the Pareto frontier. We call this approach $\boldsymbol{\alpha}$**-sweep**. Note that, the $Power$ and $Delay$ values in Eq. (8) are normalized and scaled to the range between 0 and 1 by Eq. (9).

$$x = \frac{x - \min(X)}{\max(X) - \min(X)}, x \in X. \tag{9}$$

Similarly, we have a joint output Area-Delay (AD) function for Pareto frontier exploration on Area and Delay space.

$$AD = \alpha \cdot Area + Delay. \tag{10}$$

This α-sweep technique can be extended to simultaneously consider delay, area and power, using two scalars (α_1 and α_2) instead one scalar factor α. For the sake of simplicity, we have demonstrated the methodology separately for delay vs. area and delay vs. power using one scalar at a time.

IV. Experimental Results

In this section we show the effectiveness of our algorithms and methodologies in the physical solution space. Since high performance adders are typically used in CPU architectures which are typically 64 bit, we have mainly presented the results for 64 bit adders to demonstrate the methodology. 300 solutions, sampled from all prefix adder solutions by the quasi-random approach, are used for training (250) and testing (50) the learning model. However, the approach is very general to be used for adders of arbitrary bit-width. The flow is implemented in C++ and Python on Linux machine with 72GB RAM and 2.8GHz CPU. We use Design Compiler [22] (version F-2011.09-SP3) for logical synthesis, and IC Compiler [24] (version J-2014.09-SP5-3) for the placement and routing. Non Linear Delay Model (NLDM) in $32nm$ SAED cell-library [25] (available by University Program) is used for technology mapping. Primary input activity of 0.1 is used along with 1GHz operating frequency for power estimation. Targets delays of $0.1ns$, $0.2ns$, and $0.3ns$ are used during synthesis. We used Python based machine learning package scikit-learn [26] for the predictions. Throughout our all experiments, the run time for machine learning predictions is less than a minute.

To validate the optimality of the predicted Pareto frontier by α-sweep against the real world solution space, we need to run the logical/physical EDA flow on a large set of adder solutions. Our machine and tool set takes about 6 min to complete this full flow of a single prefix adder. Therefore, we select a reasonable number (3000) of prefix adder solutions, which eventually took about 300 hours to complete, but still a comparatively larger data set in comparison to our training data set. Crucially, those 3000 adders are also sampled in a **Quasi-random** manner in order to represent the entire solution space.

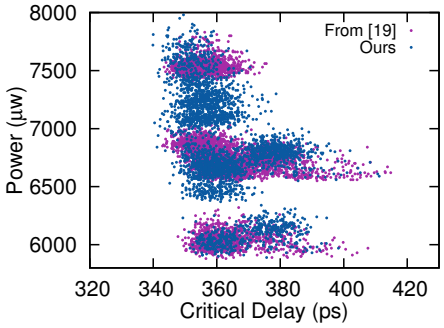
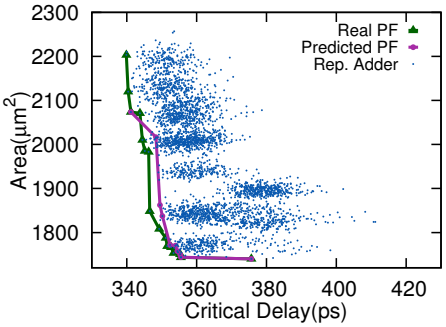
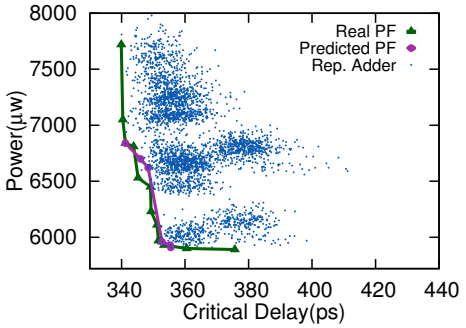


Fig. 5: Quasi-random sampled adders vs. adders from [19].



(a) Pareto frontier: delay vs. area



(b) Pareto frontier: delay vs. power

Fig. 6: Predicted Pareto frontier comparison.

A. Physical Solution Space Comparison with [19]

In our first experiment, we show the usefulness of our algorithm for obtaining wider solution space in physical design domain in comparison to [19]. Among the prefix adders generated by [19], we randomly sampled 7000 prefix adders. Those prefix adders are also exhaustively fed into the full EDA flow to get their real delay, power and area values (takes around 700 hours). We plot these adders by [19] and our representative 3000 adders by quasi-random sampling in Fig. 5. It can be seen that, although the numbers of adders by [19] is more than 2 times of our representative adders, our adders still cover *wider* solution space in physical domain, demonstrating the effectiveness of our enhancement algorithm PGG and the quasi-random sampling approach. This is in accordance with the solutions missed by [19] as mentioned in TABLE I. Those availabilities eventually offer more opportunities for our later machine learning methodology to identify close to ground truth Pareto frontier solutions.

TABLE II: Comparison with other approaches for 64 bit adders

Method	Delay (ps)	Area (μm^2)	Power (mW)
Kogge-Stone	347.9	2563.7	8.78
Ours (P_1)	340.0	2203.3	7.72
Sklansky	356.1	1792.5	6.1
Ours (P_2)	353.0	1753.0	5.9
[18]	348.7	1971.4	6.98
Ours (P_3)	346.0	1848.6	6.67

B. Validation of the Predicted Pareto Frontier

In this experiment, we show the effectiveness of our Pareto frontier driven machine learning approach. We apply the α-sweep method with 15 different α values of $(1000, 0, 100, \frac{1}{100}, 50, \frac{1}{50}, 20, \frac{1}{20}, 10, \frac{1}{10}, 8, \frac{1}{8}, 2, \frac{1}{2}, 1)$, and collect the best 150 solutions for delay-area and delay-power spaces where for each α value, the best 10 architectures with lowest PD or AD values are fed into the logical/physical EDA flow to generate similar Pareto points. Note that $15 + 15 = 30$ learning models have been derived for this for all, but it is very fast as the same training data have been used, and the models are regression based.

Fig. 6(a) and Fig. 6(b) respectively show the corresponding Pareto frontiers (PF) of the α-sweep approach and the ground truth PF for the 3000 representative adders. Each dot in the delay-area or delay-power space indicates one adder solution after going through the logical/physical EDA flow. We can see that generally the predicted PF solutions are fairly close to the real Pareto frontier, with some exceptions. Overall, the proposed approach can effectively achieve near optimal Pareto frontier without affording to spend expensive runtime on every adder. So this learning based methodology can be readily adopted to achieve Pareto frontiers for much larger solution space which is intractable for exhaustive exploration by conventional design flow.

C. Adder Performance Comparison

Finally, we compare our explored optimal adders against legacy adders, such as Kogge-Stone, Sklansky, as well as the state-of-the-art adder synthesis algorithm in TABLE II. Since our approach generates numerous solutions, it is not feasible to perform a one-to-one comparison. Instead for each of the solution points in regular adders and [18], we have picked the Pareto points from our solution set which are able to excel them in all metrics. For instance, P_1 could provide around 8ps better delay with respectively 14% and 12% lesser area and power over Kogge-Stone adder. For [18] we pick the best delay solution. Note for a fixed mfo, [18] can give prefix network with smaller size, but this approach only provides a limited set of prefix structures. As a result, it is hard for [18] to explore the full physical design space of adders by machine learning. It should be stressed that [18] beats the custom adders implemented in an industrial design, and our methodology is able to excel the adders generated by the algorithm presented in [18].

V. Conclusion

This paper presents a novel methodology of machine learning guided design space exploration for power efficient high-performance prefix adders. We have successfully demonstrated the effectiveness of our learning model, developed by training with quasi-random sampled data and features encapsulating architectural and tool attributes. Our adder synthesis algorithm is able to generate a wider solution space in comparison to a state-of-the-art algorithm, and when integrated with the learning model, could provide a near-optimal performance vs. area/power Pareto frontier over a large representative solution space. To the best of our knowledge, this is the first work to bridge the gap between architectural and physical solution space for parallel prefix adders. With increasing design complexity and cross-layer uncertainties, we anticipate that our methodology will become more and more relevant in other EDA problems.

In future we would extend our methodology to address the gap between architectural/logic stage and physical design for other circuits. However, it may be difficult always to find the regularity in the circuit structure like prefix adders. This can cause a hindrance to find the features, but in that case, multilayer neural network can be employed for efficient feature learning.

References

[1] B. Yu, D. Z. Pan, T. Matsunawa, and X. Zeng, "Machine learning and pattern matching in physical design," in *Proc. ASPDAC*, 2015, pp. 286–293.

[2] W.-T. J. Chan, K. Y. Chung, A. B. Kahng, N. D. MacDonald, and S. Nath, "Learning-based prediction of embedded memory timing failures during initial floorplan design," in *Proc. ASPDAC*, 2016, pp. 178–185.

[3] P. Meng, A. Althoff, Q. Gautier, and R. Kastner, "Adaptive threshold non-pareto elimination: Re-thinking machine learning for system level design space exploration on FPGAs," in *Proc. DATE*, 2016, pp. 918–923.

[4] W.-H. Chang, L.-D. Chen, C.-H. Lin, S.-P. Mu, M. C.-T. Chao, C.-H. Tsai, and Y.-C. Chiu, "Generating routing-driven power distribution networks with machine-learning technique," in *Proc. ISPD*, 2016, pp. 145–152.

[5] R. Samanta, J. Hu, and P. Li, "Discrete buffer and wire sizing for link-based non-tree clock networks," in *Proc. ISPD*, 2008, pp. 175–181.

[6] R. P. Brent and H. T. Kung, "A regular layout for parallel adders," *IEEE Transactions on Computers*, vol. C-31, no. 3, pp. 260–264, 1982.

[7] P. M. Kogge and H. S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Transactions on Computers*, vol. 100, no. 8, pp. 786–793, 1973.

[8] T. Han and D. Carlson, "Fast area-efficient VLSI adders," *Proc. ARITH*, pp. 49–56, 1987.

[9] J. Sklansky, "Conditional sum addition logic," *IRE Trans. on Electronic Computers*, vol. EC-9, no. 2, pp. 226–231, 1960.

[10] C. Zhou, B. M. Fleischer, M. Gschwind, and R. Puri, "64-bit prefix adders: Power-efficient topologies and design solutions," *Proc. CICC*, pp. 179–182, 2009.

[11] J. Liu, Y. Zhu, H. Zhu, C.-K. Cheng, and J. Lillis, "Optimum prefix adders in a comprehensive area, timing and power design space," in *Proc. ASPDAC*, 2007, pp. 609–615.

[12] T. Matsunaga and Y. Matsunaga, "Area minimization algorithm for parallel prefix adders under bitwise delay constraints," in *Proc. GLSVLSI*, 2007, pp. 435–440.

[13] J. Liu, S. Zhou, H. Zhu, and C.-K. Cheng, "An algorithmic approach for generic parallel adders," in *Proc. ICCAD*, 2003, pp. 734–740.

[14] J. P. Fishburn, "A depth-decreasing heuristic for combinational logic; or how to convert a ripple-carry adder into a carry-lookahead adder or anything in-between," in *Proc. DAC*, 1990, pp. 361–364.

[15] R. Zimmermann, "Non-heuristic optimization and synthesis of parallel prefix adders," *Proc. IWLAS*, pp. 123–132, 1996.

[16] M. Snir, "Depth-size trade-offs for parallel prefix computation," *Journal of Algorithms*, vol. 7, no. 2, pp. 185–201, 1986.

[17] H. Zhu, C.-K. Cheng, and R. Graham, "Constructing zero-deficiency parallel prefix adder of minimum depth," in *Proc. ASPDAC*, 2005, pp. 883–888.

[18] S. Roy, M. Choudhury, R. Puri, and D. Z. Pan, "Polynomial time algorithm for area and power efficient adder synthesis in high-performance designs," in *Proc. ASPDAC*, 2015, pp. 249–254.

[19] ——, "Towards optimal performance-area trade-off in adders by synthesis of parallel prefix structures," *IEEE TCAD*, vol. 33, no. 10, pp. 1517–1530, 2014.

[20] B. R. Zeydel, T. T. J. H. Kluter, and V. G. Oklobdzija, "Efficient mapping of addition recurrence algorithms in CMOS," *Proc. ARITH*, pp. 107–113, 2005.

[21] M. Ketter, D. M. Harris, A. Macrae, R. Glick, M. Ong, and J. Schauer, "Implementation of 32-bit ling and jackson adders," *Proc. ASILOMAR*, pp. 170–175, 2011.

[22] "Synopsys Design Compiler," http://www.synopsys.com.

[23] F. Girosi, M. Jones, and T. Poggio, "Estimating the bayes error rate through classifier combining," in *Neural Comput.*, vol. 7, 1995, pp. 219–269.

[24] "Synopsys IC Compiler," http://www.synopsys.com.

[25] http://www.synopsys.com/Community/UniversityProgram/Pages/32-28nm-generic-library.aspx, accessed 23-April-2016.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.