Graph-Based Logic Bit Slicing for Datapath-Aware Placement *

Chau-Chin Huang[1], Bo-Qiao Lin[1], Hsin-Ying Lee[1], Yao-Wen Chang[1,2], Kuo-Sheng Wu[3], and Jun-Zhi Yang[3]
[1]Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106, Taiwan
[2]Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan
[2]MediaTek, Inc., Hsinchu 300, Taiwan
{wlkb83, bqlin, hylee}@eda.ee.ntu.edu.tw; ywchang@ntu.edu.tw; {mike.wu,bauli.yang}@mediatek.com

ABSTRACT

Extracting similar datapath bit slices which handle highly parallel bit operations can help a modern placer to obtain better solutions for datapath-oriented designs. A current state-of-the-art datapath bit slicing method achieves the best extraction results using a network-flow-based algorithm. However, this work has two major drawbacks: (1) it extracts only a limited number of bit slices for datapaths with different I/O widths, which are commonly seen in real designs, and (2) it does not consider bit-slice similarity, which is an important feature for placement considering datapaths. To remedy these drawbacks, we present (1) a balanced bipartite edge-cover algorithm to fully slice a datapath with different I/O widths, and (2) a simulated annealing scheme to further improve bit-slice similarity, while maintaining fully-sliced structures. Compared with the state-of-the-art work, experimental results show that our slicing algorithm extracts more bit slices with similar structures, and helps a leading academic placer achieve averagely 5% smaller routed wirelength. The results also validate the high correlation between datapaths and structure regularity/similarity.

1. INTRODUCTION

A datapath contains an array of *bit slices*, which transfers multiple-bit data from a primary input (PI)/latch vector to a primary output (PO)/latch vector. In a datapath, a bit slice is defined as a set of gates manipulating a bit operation from one PI/PO/latch to another. Figure 1(a) shows a datapath with three bit slices, where *Bit slice 1* has eleven datapath gates, excluding I/Os.

Because the array of datapath bit slices transfers multiple-bit data and performs parallel bit operations, the slices are likely to share similar/regular structures [11]. Nijssen *et al.* [12] pointed out that the regular structures in a datapath can help to achieve compact placement and meet modern aggressive design constraints [2, 6, 7, 8, 9, 17]. (Certainly, the correlation between such regularity and a real datapath is not 100%, but it is sufficiently high to achieve significantly better placement solutions; for example, it will be clear in Section 4 that our algorithm can obtain averagely 8% smaller routed wirelength based partly on structure regularity and similarity.) However, directly extracting regular structures of a datapath is often difficult for modern digital designs mainly because the parallel bit operations may be synthesized to different structures after synthesis or technology mapping [13, 19]. As shown in Figure 1(a), it is hard to identify the regularity for the three bit slices which are functionally identical, but structurally different.

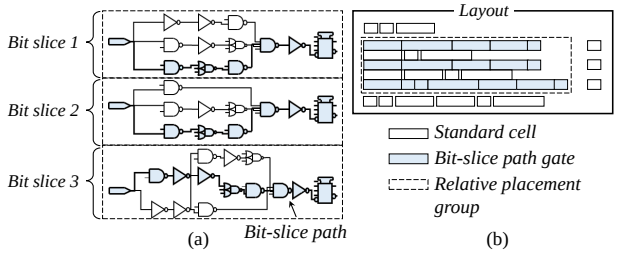


Figure 1: (a) A datapath with three bit slices performing the same function, which have similar but different structures. (b) Aligning the three highlighted bit-slice paths in (a) may help to achieve high utilization and good routability. Note that this is a sub-circuit of an industrial design extracted from [19].

Instead of stressing on regularity extraction alone, this paper focuses on identifying *datapath bit-slice paths*—a set of bit lines which are *more similar and longer* than any other set of bit lines in the bit slices. As shown in Figure 1(a), we intend to extract the highlighted bit line (bit-slice path) for each bit slice. Because these bit-slice paths are more similar and longer than others, one widely adopted placement strategy is to align extracted bit-slice paths toward the same direction [5, 6]. Ward *et al.* [18] further pointed out that other gates in each bit slice become aligned as well during an iterative placement flow. This alignment of highly parallel bit operations can effectively reduce corresponding Steiner wirelength and routing congestions [18]. Therefore, we follow these guidelines and intend to extract a set of bit-slice paths which provide an alignment guidance to a datapath-aware placer in this paper. For example, Figure 1(b) shows a resulting placement by aligning the three highlighted bit-slice paths from left to right in a relative placement group [1] [15].

The basic idea of the bit-slice paths is analogous to that of *datapath main frames* in [19] *(the ISPD'13 best paper)*, which provides the same alignment guidance. Datapath main frames refer to a set of *disjoint* bit lines such that the number of datapath gates on these extracted bit lines is maximized. However, main frames in [19] may not provide a complete alignment guidance for datapaths with different I/O widths. For example, Figure 2(a) illustrates a simple 4-to-2 datapath, which manipulates arithmetic operations between 2-bit input vectors A, B, and an output vector X. As shown in Figure 2(b), the network-flow-based method can only identify two "disjoint" main frames: $A1 \rightsquigarrow X0$ and $B1 \rightsquigarrow X1$ (with red arrows), due mainly to the unit edge capacity setting of the network-flow-based method. If we directly align these two main frames, the other two input bits $(A0, B0)$ and the corresponding parallel bit operations would not be considered, which may result in an inferior datapath placement solution. In contrast, Figure 2(d) shows our slicing result with four bit-slice paths, where all bits in the I/O vectors are covered by bit-slice paths, revealing that our slicing result can provide better alignment information to datapath placement.

One intuitive way to extend the network-flow-based extraction method

*This work was partially supported by AnaGlobe, MediaTek, RealTek, TSMC, MOST of Taiwan under Grant NSC 102-2221-E-002-235-MY3, NSC 102-2923-E-002-006-MY3, MOST 103-2221-E-002-259-MY3, MOST 103-2812-8-002-003, MOST 104-2221-E-002-132-MY3, MOST 103-2923-E-002-011-MY3, MOST 102-2221-E-002-136-MY3, and NTU under Grant NTU-ERP-105R8951.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DAC '17, June 18-22, 2017, Austin, TX, USA
Copyright 2017 ACM 978-1-4503-4927-7/17/06$15.00
http://dx.doi.org/10.1145/3061639.3062254 ...$15.00.

[1]The relative placement group technique is supported by IC Compiler™. It controls the relative placement topology of gate-level logic group.

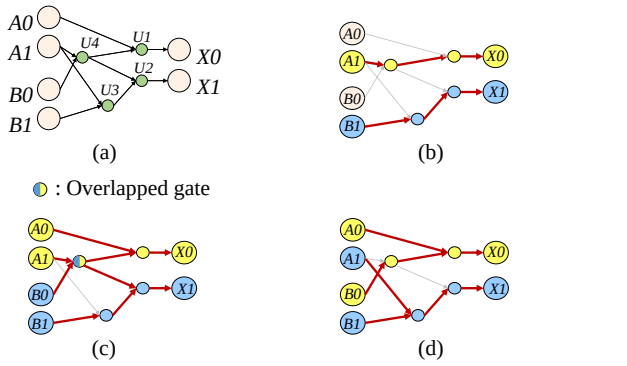


Figure 2: Comparison of bit slicing algorithms. (a) A datapath circuit with 4-to-2 I/O widths. (b) A network-flow-based bit slicing solution [19] with only two I/O bit pairs: $A1 \rightsquigarrow X0$ and $B1 \rightsquigarrow X1$. (c) A possible bit slicing solution produced by the two extensions of the network-flow-based algorithm in [19], which contains an overlapped gate. (d) Our *fully-sliced* bit slicing solution covering all I/O bits: $\{A0, B0\} \rightsquigarrow \{X0\}$ and $\{A1, B1\} \rightsquigarrow \{X1\}$, which contains similar two bit slices and no overlapped gate.

to completely cover all the PIs/POs/latches is to iteratively apply this algorithm to the remaining circuit with unprocessed PIs/POs/laches. However, it would result in many overlaps among extracted bit slices. For example, Figure 2(b) shows the network-flow-based result for the first iteration. For the second iteration, because $\{A1, B1\}$ were processed and thus excluded, only the unprocessed main frame $\{A0, B0\}$ is left to be chosen. After two iterations, we would obtain a slicing result with two bit slices: $\{A0, A1\} \rightsquigarrow \{X0\}$ and $\{B0, B1\} \rightsquigarrow \{X1\}$ as shown in Figure 2(c). Another extension is to increase the node capacities to cover all the PIs/POs/latches. For example, we could set the node capacities for $U1, U2, U3, U4, X0$, and $X1$ to two, but it may also lead to the solution with an overlapped gate shown in Figure 2(c). Because the two network-flow-based extensions above are hard to minimize the number of overlapped gates between bit slices, it may fail to provide a good alignment guidance to datapath placement.

1.1 Our Contributions

No previous works can extract similar bit-slice paths for a datapath with different I/O widths. To tackle this problem, this paper formulates a bit-slice path extraction problem and solves it with an edge cover algorithm and a simulated annealing scheme. We summarize the key contributions of this paper as follows.

- We model the original datapath circuit into a bipartite graph, where each bipartite edge represents a potential bit-slice path. This modeling significantly reduces the problem size of the following edge-cover algorithm.
- We apply a balanced bipartite edge-cover algorithm which guarantees to completely slice a datapath with *different I/O widths*. As shown in Figure 2(c), we completely slice the 4-to-2 datapath into two 2-to-1 bit slices.
- We propose a simulated annealing scheme to optimize bit-slice similarity, where three types of moves are employed to effectively refine the slicing solution. An incremental update technique is further proposed to speed up the annealing process.
- Experimental results show that our algorithm can find similar slicing structures for a datapath even with different I/O widths, while the network-flow-based algorithm cannot. Further, our extracted bit-slice paths lead to placements with smaller routed wirelength than the previous work.
- Our results also validate the high correlation between datapaths and structure regularity/similarity.

The remainder of this paper is organized as follows. Section 2 introduces bit-slice paths and formulates the problem. Section 3 presents our datapath extraction algorithm. Experimental results are given in Section 4, followed by the conclusion in Section 5.

2. PRELIMINARIES

In this section, we introduce the properties and feasibility of bit-slice paths and then define a new bit-slicing problem which handles datapaths with different I/O widths.

2.1 Bit-Slice Path Properties and Feasibility

We summarize the properties of bit-slice paths as follows:

1. Bit-slice paths are similar (with a small area variance);
2. Bit-slice paths are long (with a large area mean);
3. The overlaps between bit-slice paths is minimized.

The first property indicates that the paths may perform similar bit operations, and aligning these paths allows multiple-bit data to be transferred more regularly. The second property implies that longer similar paths are better than shorter ones, providing a bit-slice alignment guidance. The third property guides bit slice placement with higher flexibility by allowing overlapped gates between the extracted bit-slice paths [2]. Note that our extraction algorithm considers these three properties simultaneously with a weighting scheme. For a datapath which lacks some properties inherently (e.g., an adder datapath with a carry chain lacks the similarity property of each bit slice), our algorithm can still provide a placement-friendly result by tuning up the weights of other properties.

Bit-slice paths feasibility ensures that all datapath I/O bits are covered by the extracted bit slices, providing complete alignment information to datapath placement. We first give the definitions of a *complete slicing solution* and a *feasible* set of bit-slicing paths, and then explain the implications of these two definitions.

DEFINITION 1. *A slicing solution is said to be* complete *if every bit in a set of I/O vectors is assigned to a bit slice.*

Let a *wide (narrow) vector bit* represent a bit in the I/O vector with a larger (smaller) width.

DEFINITION 2. *A feasible set of bit-slice paths is a set of overlap-allowed bit-slice paths satisfying the following two properties:*

- *Each wide vector bit connects to exactly one bit-slice path.*
- *Each narrow vector bit connects to at least one bit-slice path.*

For Definition 1, we intend to consider all I/O bits for better alignment information. According to Definition 2, a complete slicing solution can be induced from a feasible set of bit-slice paths as follows. In a feasible set of bit-slice paths, each bit in the given I/O vectors has at least one bit-slice path connecting to it. For each bit in the narrow I/O vector, we regard it as the ending bit of an extracted bit slice. With bit-slice paths, we select corresponding bits in the wide I/O vector as the ending bits of the bit slice. Thus, every bit in the I/O vectors is assigned to a bit slice, and the number of bit slices equals the width of the narrow vector. Figure 2(d) shows a complete slicing solution which is also a feasible set of bit-slice paths. There are four bits in the wide vector and two bits in the narrow vector. The bit slice $\{A0, B0\} \rightsquigarrow \{X0\}$ consists of two bit-slice paths (yellow), and the bit slice $\{A1, B1\} \rightsquigarrow \{X1\}$ consists of two bit-slice paths (blue). Therefore, we have the following theorem:

THEOREM 1. *Given an arbitrary datapath (with different or identical I/O widths), every feasible set of bit-slice paths corresponds to a complete bit slicing solution.*

Furthermore, because the problem scope in [19] is limited to datapaths with identical I/O widths, we give Definition 2 for bit-slice paths feasibility to capture the general datapath slicing problem precisely.

2.2 Problem Formulation

Because setting an attribute on PI/PO/latch in a design is simple, datapath I/O vectors can be easily obtained in the physical design stage [19]. Further, because these datapath I/O vectors divide circuit sequential loops into several combinational sub-circuits, we can identify *datapath gates* for every sub-circuit by a two-way search method

[2]Ideally, bit-slice paths should be disjoint paths to enable a separate aligning placement. However, overlaps between multiple-bit data are inevitable in most real designs. Therefore, it is desired to minimize the number of overlapped gates.

proposed in [19] according to the corresponding I/O vectors. That is, each of these gates has at least one path connecting to the input vector and to the output vector. Because these gates have potential to be part of bit-slice paths, we focus on extracting bit-slice paths among these datapath gates. The problem is defined as follows, and Figure 3 lists the notations used in this paper.

PROBLEM 1 (BIT-SLICE PATH EXTRACTION). *Given a datapath D with N gates, M nets, an l-bit wide vector L, and an s-bit narrow vector S, $l \geq s$, identify a feasible set of bit-slice paths P such that (1) the gate area variance of P is minimized, (2) the gate area mean of P is maximized, and (3) the number of overlapped gates in P is minimized.*

D	the given datapath
N	the number of gates in D
M	the number of nets in D
L	the wide (**L**arge) I/O vector, where $L = (l_1,...,l_l)$
S	the narrow (**S**mall) I/O vector, where $S = (s_1,...,s_s)$
l, s	the widths of L and S, where $l \geq s$
p_{ij}	the bit-slice path to l_i and s_j, where $i \leq l$, $j \leq s$
P_i	the set of bit-slice paths connecting to s_i, where $i \leq s$
P	the set of bit slice paths P_i for $1 \leq i \leq s$

Figure 3: Notations in this paper.

3. ALGORITHMS

The objective of the bit-slice path extraction problem is to find a set of bit lines with a small area variance, a large area mean, and a small number of overlapped gates while satisfying the feasibility of bit-slice paths. It is hard to solve this multi-objective problem directly and simultaneously, which motivates us to resort to a divide-and-conquer method with two stages: (1) we first optimize the area variance and area mean with a balanced bipartite edge-cover algorithm; (2) we then minimize the number of overlapped gates with a simulated annealing scheme while preserving the qualities of the area variance and mean. Figure 4 summarizes our proposed algorithm.

3.1 Balanced Bipartite Edge-Cover Algorithm

Based on the observation that every bit-slice path is a bit line connecting I/O vectors, it is intuitive to model the datapath D as a bipartite graph $G = (V_L, V_S, E)$, where the vertex sets V_L and V_S correspond to the wide (larger) I/O vector L and narrow (smaller) I/O vector S, respectively. The edge set E corresponds to a set of bit lines in D. That is, we transform the problem of extracting a feasible set of bit-slice paths in D into a problem of covering all vertices in G.

3.1.1 Bipartite Graph Construction

For the bipartite graph G, each edge has an associated weight, given by a weight function $w_e : E \to \mathbb{R}$. Because we intend to extract bit-slice paths with a large area mean, we define the weight $w_e(i,j)$ of edge $(i,j) \in E$ as the total gate area of the bit line with the *maximum* area from l_i to s_j in the datapath D.

To model the datapath D with the bipartite graph G, we first transform D into a weighted directed graph $G_D = (V_D, E_D)$ with the weight function $w_D : V_D \to \mathbb{R}$ mapping vertices to the corresponding gate areas, where the vertex set V_D corresponds to the set of N gates and the edge set E_D corresponds to the set of decomposed two-pin nets of M nets in D. The weight $w_{path}(p)$ of the simple path p is the sum of the weights of its constituent vertices:

$$w_{path}(p) = \sum_{\forall v \in p} w_D(v). \tag{1}$$

The weight $w_e(i,j)$ equals 0 if there is no path from v_i to v_j; otherwise, it is defined by

$$w_e(i,j) = \max\{w_D(p) : p \text{ is a path from } v_i \text{ to } v_j\}. \tag{2}$$

Because the datapath circuit D contains no combinational loop, the edge weight of the bipartite graph G can be obtained by optimally solving the single-source longest-path problem [14] in a directed acyclic graph for each narrow vector bit in $O(V_D + E_D)$ time. As shown in Figure 2(a), the lower directed acyclic graph can be transformed into a bipartite graph as that in Figure 5(a), where each bipartite edge weight corresponds to the area of the bit line of a datapath.

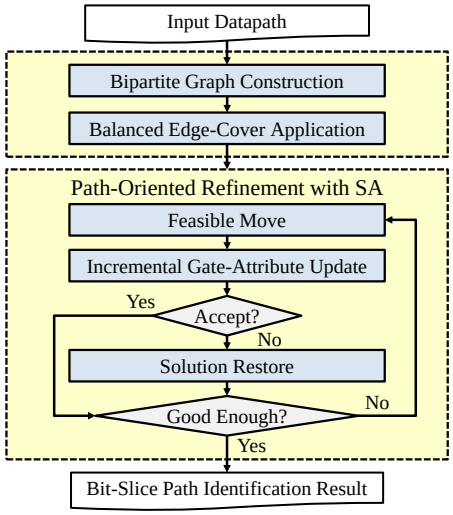


Figure 4: Overview of the bit-slice path extraction algorithm.

Algorithm 1 Balanced_Bipartite_Edge_Cover

Input: $G = (V_L, V_S, E)$
Output: Balanced edge cover $E' \subseteq E$
1 $E' \leftarrow M \leftarrow$ MaximumMatching(G)
2 **for** each $v \in V_L$
3 **if** v is not M-saturated
4 choose $e \in E$ as a maximum-weight edge on v
5 $E' \leftarrow E' \cup \{e\}$
6 **for** each $v \in V_S$
7 calculate the total incident-edge weight $w(v)$
8 **while** variance of vertex weights for V_S can be reduced
9 choose $v \in V_S$ with the largest weight such that there are at least two edges in E' incident on v
10 **if** v doesn't exist
11 **break**
12 $v_{min} \leftarrow \phi$
13 **for** each $(u,v) \in E'$
14 **for** each $(u,v') \in E \setminus E'$ with $w(v) > w(v')$
15 **if** $v_{min} = \phi$ or $w(v') < w(v_{min})$
16 $v_{min} \leftarrow v'$, $u' \leftarrow u$
17 **if** $v_{min} \neq \phi$
18 $E' \leftarrow E' \cup \{(u', v_{min})\} \setminus \{(u', v)\}$
19 update $w(v)$ and $w(v_{min})$
20 **return** E'

3.1.2 Balanced Edge-Cover Application

To obtain feasible bit-slice paths, we first apply a minimum bipartite edge cover algorithm. To further minimize the area variance of bit-slice paths, we then adopt an iterative balancing technique while guaranteeing the feasibility. Algorithm 1 summarizes the application. Lines 1–5 and Lines 8–19 give the edge cover process and iterative balancing techniques, respectively. Lines 6–7 preprocess the necessary information for Lines 8–19. In the edge cover process, Line 1 calls a maximum cardinality matching algorithm to obtain a matching solution M. Lines 2–5 select one edge for each *M-unsaturated vertex* to obtain a minimum edge cover E', where an M-unsaturated vertex refers to a vertex not covered by the matching M.

The iterative balancing technique further minimizes the area variance of bit-slice paths. We define the weight $w(v)$ of vertex $v \in V_S$ as the total incident-edge weight for an edge cover E':

$$w(v) = \sum_{i=1}^{k} w_e(u_k, v), \tag{3}$$

where v is incident on k different edges in E'. Therefore, $w(v)$ represents the total area of extracted bit lines covering v. Lines 8–19 iteratively select two edges (u', v) and (u', v_{min}) for swapping, where $v, v_{min} \in V_S$ and $u' \in V_L$. The goal of the swapping process is to reduce the difference

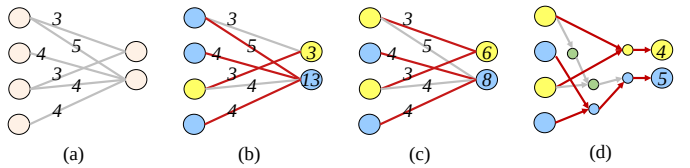


Figure 5: Illustration of the bipartite edge cover algorithm for the datapath in Figure 2. (a) The derived bipartite graph, where the edge weight corresponds to the area of the bit line (assuming each gate area equals one). (b) An initial edge cover, where the number in each output bit denotes the total area of its bit-slice paths. (c) A balanced edge cover. (d) The slicing result of (c) being mapped to the original directed acyclic graph.

of $w(v)$ and $w(v_{min})$. The process in Lines 8–19 terminates when the variance of vertex weights for V_S can no longer be reduced.

Figure 5 illustrates the balanced bipartite edge-cover algorithm on the datapath in Figure 2. Figure 5(a) is the corresponding bipartite graph. Figures 5(b) and (c) are the edge cover results before and after the balanced edge-cover application, respectively. The number in each output bit (i.e., $w(v)$) represents the total incident edge weights of the corresponding edge cover, which is also the corresponding bit-slice path area. The variance of the numbers in Figure 5(c) is effectively reduced compared with that in Figure 5(b). Figure 5(d) shows the final result of Figure 5(c) on the directed acyclic graph. Note that because the bipartite graph cannot accurately model the overlapped gates in each bit line, the number in each output bit of Figure 5(d) is different from that of Figure 5(c). Thus, a refinement technique which can effectively reduce the overlapped gates is presented in the following subsection.

THEOREM 2. *Balanced_Bipartite_Edge_Cover guarantees to find a complete slicing solution in $O(E \cdot (V_L + V_S + k))$ time if such a solution exists, where k is the iteration number for the* `while` *loop.*

3.2 Path-Oriented Refinement with Simulated Annealing

One intuitive way to find a desired feasible solution for the bit-slice path extraction is to enumerate all possible bit-slice paths. However, an exhaustive technique may not be suitable for this problem because even the number of possible single-source paths in a graph could be exponential to that of the vertices [3], let alone that the solution of our problem is a set of paths, not only one path. Consequently, we resort to simulated annealing (SA) [4] for better speed and quality trade-offs. In the following four subsections, we study the properties of the bit-slice path extraction problem and detail an SA framework for the problem.

3.2.1 Solution Representation

A *feasible solution* for SA refers to a set of simple directed paths P_D in a weighted directed graph G_D if and only if P_D corresponds to a feasible set of bit-slice paths in D. Our best-evaluated feasible solution coincides with an optimal solution of the original bit-slice path extraction problem because the transformation of two instances in D and G_D is a 1-to-1 correspondence and the area of a gate for D and the corresponding vertex weight for G_D are equal.

3.2.2 Neighborhood Structure

We define three moves to update a solution during SA.

- M1: Delete and insert a bit-slice path for a bit $l_i \in L$.
- M2: Reconstruct a bit-slice path for a bit $s_i \in S$.
- M3: Swap two bit-slice paths, one from bit s_i and the other from bit s_j, where $s_i, s_j \in S$, and $s_i \neq s_j$.

Figure 6(a) shows an initial solution of the bipartite edge cover algorithm. Figure 6(b) shows a resulting solution after an M1 move on bit B. The original bit-slice path $B \rightsquigarrow b$ is deleted and a new bit-slice path $B \rightsquigarrow a$ is inserted. Figure 6(c) shows a resulting solution after an M2 move on bit d. The original bit-slice path of bit d is deleted and a new path $D \rightsquigarrow d$ is randomly chosen. Figure 6(d) shows a resulting solution after an M3 move on bits e and f. We swap the original bit-slice paths $E \rightsquigarrow e$ and$F \rightsquigarrow f$ with new bit-slice paths $E \rightsquigarrow f$ and $F \rightsquigarrow e$ [3].

Figure 6 demonstrates that our refinement algorithm can achieve good bit-slice similarity. The bit-slice paths areas $(4, 10, 6, 7, 4)$ in Figure 6(a) are changed to $(5, 5, 5, 5, 5)$ in Figure 6(d).

We maintain the feasibility of solutions during the SA search process. If a move leads to an infeasible solution, the move will be discarded. The following theorem claims that it takes only constant time to check whether a move will lead to an infeasible solution.

THEOREM 3. *Given a feasible solution and a move, whether the move will lead to an infeasible solution can be checked in constant time.*

3.2.3 Cost Function

Given a set of bit-slice paths P in a datapath D, we minimize the cost function c:

$$c = \alpha \times (\hat{\mu} - \mu) + \beta \times \sigma + \gamma \times o, \tag{4}$$

where μ, σ, and o denote the area mean, the area standard deviation, and the number of overlapped gates of P, respectively. We denote $\hat{\mu}$ as the upper bound of μ:

$$\hat{\mu} = \frac{1}{s} \cdot \sum_{g=1}^{N} a_g, \tag{5}$$

where s denotes the number of bit-slices in the datapath D, a_g the area of gate g in D, and α, β, and γ user-specified parameters. In our experiments, setting $\alpha = 0.1$, $\beta = 0.5$, and $\gamma = 5$ gives satisfactory results. The idea is to focus on minimizing the overlapped number, while maintaining the quality with the other two objectives. Please note that we first randomly produce feasible solutions as profiling to obtain the approximate averages of the three costs. Then, each cost is normalized by its average.

In SA, we need to compute the cost many times. Since each move is only a local perturbation of the current set of bit-slice paths, many bit-slice paths remain unchanged, and typically only one or two bit-slice paths need to be recomputed. Thus, the annealing process can be significantly speeded up if we can incrementally update the three costs μ, σ, and o. We denote the corresponding total bit-slice paths area before and after a move as A and A', respectively. The incremental update for the area means μ' of bit-slice paths after a move:

$$\mu' = \mu + \frac{A' - A}{s}. \tag{6}$$

The incremental update of the area standard deviation σ' of bit-slice paths after a move:

$$\sigma' = \sqrt{\sigma^2 - (\frac{A' - A}{s})^2 + \frac{A'^2 - A^2}{s} - 2\mu\frac{A' - A}{s}}. \tag{7}$$

In Equations (6) and (7), only A' is unknown for the computation, where A' can be updated by adding (subtracting) the corresponding gate area from A when a bit-slice path is inserted (deleted). Therefore, after each move, it takes only constant time to obtain the corresponding bit-slice paths area A'. The resulting number of overlapped gates o can be obtained in constant time by the same way. Thus, we have the following theorem.

THEOREM 4. *After any move of type M1, M2, or M3, it takes only constant time to evaluate the cost in Equation* (4) *for a set of bit-slice paths.*

3.2.4 The Annealing Schedule

To reduce the iterations for exploring initially inferior solutions and to spend more time on finding better solutions in the hill-climbing stage, we apply fast simulated annealing for solution refinement [16]. The refinement starts with a feasible initial solution produced by the bipartite edge cover algorithm. At each temperature, we try a sufficient number of feasible moves until either the number of uphill moves exceeds N or the total number of moves exceeds $2N$, where N is the number of gates in the datapath D. We terminate the annealing process if the rejection ratio in the current temperature is higher than 90% or the current temperature is lower than a cooling temperature $T_c = 0.1$.

[3] Note that the three moves are selected with the same probability in our experiments.

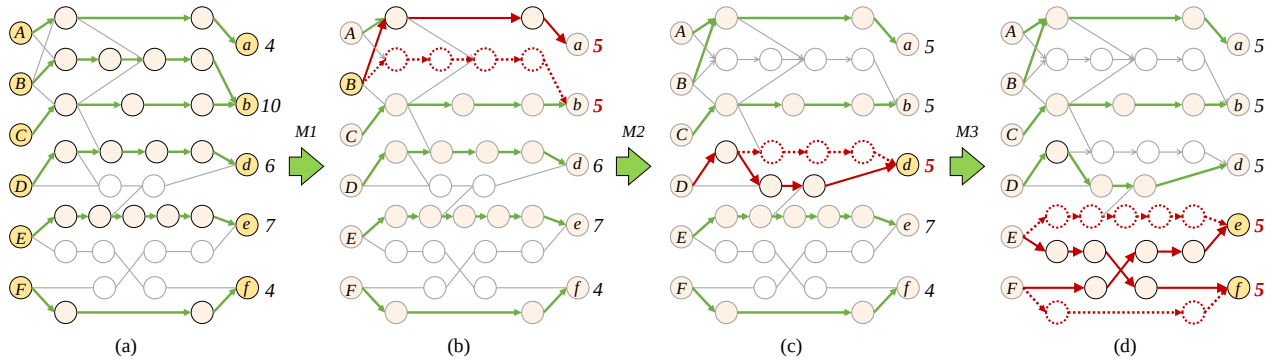


Figure 6: Three types of moves in simulated annealing. The number beside each output bit represents the corresponding total bit-slice area, assuming that the area of each gate equals one. (a) A bit-slice paths solution of the bipartite edge cover algorithm. (b) A solution after M1 move (delete and insert the path) on bit B in the wide vector. (c) A solution after M2 move (reconstruct the path) on bit d in the narrow vector. (d) A solution after M3 move (swap the paths) on bits e and f in the narrow vector. The three types of moves effectively reduce the area variance of bit-slice path: after M1, M2, and M3 moves, the bit slices areas are changed from (4, 10, 6, 7, 4) to (5, 5, 5, 5, 5).

4. EXPERIMENTAL RESULTS

The proposed extraction algorithm was implemented in the C++ programming language. Also, we implemented the network-flow-based algorithm in [19] for comparison, and compared the algorithms on five industrial datapath-intensive designs (including MIPS and LZSS designs) on a Linux workstation with Intel Xeon E5506 2.13GHz CPU and 16GB memory [4].

We conducted experiments on two testcase sets. The first set in Table 1 contains four testcases with same I/O widths. The second set in Table 2 contains 12 testcases with both same and different I/O widths. Each testcase is named as *testname-dpIndex*. For example, *TestA-2* is referred to the second sub-circuit in *TestA*.

Table 1: Bit Slicing Tests on Datapaths with Same I/O Widths ("N/A": incomplete slicing solutions).

Design Information		Slices		MinSize-BitSlice		MaxSize-BitSlice	
Datapath	Width	[19]	Ours	[19]	Ours	[19]	Ours
TestA-1	4	4	4	4	4	8	8
TestB-1	32	32	32	5	5	7	7
TestC-1	32	32	32	4	4	6	6
TestC-2	32	2	32	N/A	64	N/A	77

4.1 Slicing Comparisons for Datapaths with Same Widths

Table 1 compares the slicing results of the proposed balanced edge cover algorithm with those of the network-flow-based algorithm in [19], where "Width" gives the number of datapath I/O bits, "Slices" gives the number of bit slices obtained by the algorithms, and "MinSize-BitSlice" and "MaxSize-BitSlice" give the respective minimum and maximum numbers of gates in one slice. We do not report runtimes because they are all negligible (less than 1 second). In our experimental settings for the two algorithms, if a gate is covered by multiple bit slices, the gate belongs to the slice with the smallest number of gates. *Note that we use the same cost metrics in [19] for fair comparison.*

The experimental results show that the slicing solutions of the network-flow-based algorithm is comparable to ours for the first three testcases in Table 1 (i.e., each input bit matches the same output bit in the results of the two algorithms). However, for *TestC-2*, the network-flow-based algorithm could not obtain a complete slicing result. Here, *TestC-2* is an ALU datapath in the MIPS processor for executing 11 types of arithmetic operations. We observed that the main reason for the incomplete slicing is that the datapath does not contain 32 disjoint paths connecting among the I/O vectors. In contrast, our edge cover algorithm still found a bit slice for each bit in the I/O vectors with the number of gates in a slice ranging from 64 to 77.

[4]Note that the binary code and test cases of [19] are not available to the public.

4.2 Slicing Comparisons for Datapaths with Different Widths

Table 2 evaluates the effectiveness of our algorithm and the network-flow-based one by comparing the three objectives: the bit-slice paths area mean, bit-slice path area variance, and number of overlapped gates between bit-slice paths. In Table 2, "Inst-Num" gives the number of gates in the datapath, "Mean-Bit-Slice-Paths" gives the mean area of the resulting bit-slice paths, "Variance-Bit-Slice-Paths" gives the area variance of the resulting bit-slice paths, "Overlap-Gate-Num" gives the number of overlapped gates between the resulting bit-slice paths, "BEC" gives the slicing results of the balanced bipartite edge cover algorithm, "[19]+RSA" and "BEC+RSA" give the respective slicing results of the SA-based refinements by using the results from the network-flow-based algorithm and the "BEC" results as initial solutions, "Runtime" gives the total runtime for the results in "[19]+RSA" and "BEC+RSA", and "Average" gives the average of the normalized value based on the result in "BEC+RSA". The normalized number equals one if any of the denominators or numerators is zero.

In Table 2, we denote incomplete slicing solutions by "N/A"; the incomplete ones may misguide datapath placement. The number of the bit slices for a complete slicing solution is the same as the width of the narrow I/O vector, so the number of bit slices is not shown in Table 2. The results show that the network-flow-based algorithm may extract only a limited number of bit slices (with even incomplete ones). For the six testcases where the network-flow-based algorithm is applicable, the variance and overlap gate number can be further reduced by our refinement scheme "RSA", which shows the effectiveness of our SA scheme. The results of "BEC" and "BEC+RSA" show that "RSA" achieves an averagely 5.19X smaller area variance and 7.23X improvement of the number of overlapped gates between bit-slice paths. Please note that because "RSA" keeps the feasibility invariance, an initial solution from "BEC" ensures a feasible final slicing solution. On the other hand, "[19]+RSA" could result in an infeasible slicing solution.

4.3 Slicing Comparisons on Placement

To further examine the effects of datapath extraction on placement, we compared the following three placement flows: (1) without bit-slicing consideration, (2) with main frame consideration [19], and (3) with our bit-slice path consideration. The three flows were all based on a leading academic analytical placer [10] and routed by SoC Encounter [1]. To consider bit-slicing alignment information in placement, we augmented the placer with a simple datapath-aware placement algorithm as follows: According to the extraction result, we define a bit slice as a main frame or a bit-slice path set, which connects to a narrow vector bit. Each bit slice is then aligned horizontally at the beginning of the placement. When the objective cost of the analytical placement cannot be improved, the y-coordinate (x-coordinate) of every aligned bit slice is dynamically determined by the geometric mean (minimum) of related gates of the current placement solution. Therefore, during the analytical placement, gates in bit slices are movable and neighboring cells can honor the alignment information. Table 3 shows the statistics of the benchmarks and placement results. *Note that we excluded*

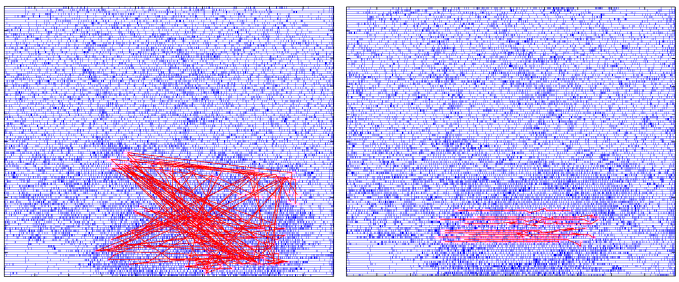
Table 2: Bit Slicing Tests on Industry Designs; BEC and RSA denote the balanced bipartite edge cover and the SA-based refinement, respectively ("N/A": incomplete slicing solutions).

Design Information			Mean-Bit-Slice-Paths				Variance-Bit-Slice-Paths				Overlap-Gate-Num				Runtime (sec)	
Datapath	I/O Widths	Inst-Num	[19]	[19]+RSA	BEC	BEC+RSA	[19]	[19]+RSA	BEC	BEC+RSA	[19]	[19]+RSA	BEC	BEC+RSA	[19]+RSA	BEC+RSA
TestA-2	8/4	25	N/A	N/A	514.0	514.0	N/A	N/A	38.0	38.0	N/A	N/A	0	0	<0.10	<0.10
TestB-2	32/32	108	413.2	510.7	656.2	513.0	101.0	35.8	39.0	33.5	0	0	30	0	<0.10	<0.10
TestC-3	64/32	1574	N/A	N/A	1046.7	996.2	N/A	N/A	76.4	30.5	N/A	N/A	40	0	N/A	72.26
TestC-4	32/32	1925	N/A	N/A	1280.0	508.7	N/A	N/A	502.3	45.7	N/A	N/A	75	8	N/A	88.04
TestC-5	32/32	66	368.5	368.5	368.5	368.5	2.7	2.78	2.7	2.7	0	0	0	0	<0.10	<0.10
TestC-6	32/32	1670	669.0	758.5	808.0	749.7	81.0	21.2	55.3	22.8	0	0	30	0	19.15	19.80
TestD-1	16/18	66	N/A	N/A	484.0	484.0	N/A	N/A	190.8	190.8	N/A	N/A	0	0	N/A	<0.10
TestD-2	84/64	259	N/A	N/A	511.1	510.6	N/A	N/A	161.1	161.2	N/A	N/A	0	0	N/A	<0.10
TestD-3	18/18	68	456.0	456.0	450.2	450.2	39.7	39.7	114.5	114.5	0	0	0	0	<0.1	<0.10
TestE-1	32/32	963	930.7	822.2	3107.2	827.0	321.3	40.4	2680.1	101.0	0	0	41	0	6.39	6.52
TestE-2	32/32	508	846.7	808	1137.5	889.7	988.9	269.8	440.3	389.4	0	0	21	0	1.80	1.83
TestE-3	64/32	1111	N/A	N/A	4181.7	1680.2	N/A	N/A	2338.3	185.8	N/A	N/A	56	11	N/A	15.73
Average	–	–	N/A	N/A	1.54	1.00	N/A	N/A	5.19	1.00	N/A	N/A	7.23	1.00	–	–

Table 3: Bit Slicing placement statistics and placement result comparisons.

	Bit-Slicing Placement Statistics				Without Bit-Slicing Info.			With Main Frame [19]			With Bit-Slice Path (Ours)		
Placement Design	#Cells	#Nets	#Pins	#Terminals	HPWL (×e4)	rWL (×e4)	CPU (sec)	HPWL (×e4)	rWL (×e4)	CPU (sec)	HPWL (×e4)	rWL (×e4)	CPU (sec)
TestB	345	336	1705	107	0.79	8.96	<1.00	0.80	8.87	<1.00	0.79	8.34	<1.00
TestC	19344	8984	68716	1160	7.75	72.71	39	7.46	71.02	55	6.80	65.77	58
TestD	257848	275320	1524888	26072	195.75	2394.00	393	185.66	2282.00	379	175.96	2170.00	679
TestE	385728	477696	2677952	39002	286.97	3287.00	726	282.39	3227.00	783	279.40	3184.00	1099
Average	-	-	-	-	1.07	1.08	0.73	1.04	1.05	0.80	1.00	1.00	1.00

TestA *in this placement experiment because it contains only 42 gates.* Table 3 summarizes the experimental results, where HPWL, routed wirelength (rWL), and placement runtime are compared. As shown in Table 3, the flow with our bit-slice paths achieves averagely 8% and 5% smaller routed wirelength, compared with Flows (1) and (2), respectively. Therefore, compared with the main frame in [19], which may contain incomplete slicing information, our bit-slice path can provide better alignment guidance to a datapath-aware placer and achieve shorter routed wirelength, at reasonable runtime overheads. *The results well validate the high correlation between datapaths and structure regularity/similarity.* Figure 7 shows two placement results on *TestC*.



(a) (b)

Figure 7: The impact of our algorithm on placement quality, based on the circuit *TestC*. Ten bit slices, which are a subset of our extraction results, are marked in red in these two placements. (a) Placement without bit-slice path consideration; (b) placement with bit-slice path consideration.

5. CONCLUSIONS

Datapath extraction is an important, yet complex process to achieve high utilization and performance. The state-of-the-art work employs a network-flow-based algorithm to tackle this problem, which incurs two major drawbacks: (1) it may generate incomplete slicing solutions, and (2) it does not consider similarity. To overcome these drawbacks, we have first defined bit-slice paths, which can provide an excellent alignment guidance to placers. We have further proposed (1) a balanced edge-cover algorithm to completely slice any datapath, and (2) an SA-based refinement process to further optimize bit-slice similarity. Experimental results show that our algorithm can extract a significantly larger number of bit slices with similar structures and obtain placements with smaller routed wirelength than the network-flow-based work. The results have also well validated the high correlation between datapaths and structure regularity/similarity.

6. ACKNOWLEDGEMENTS

We would like to thank the reviewers for their careful reviews and valuable suggestions on this paper.

7. REFERENCES

[1] *Cadence. SoC Encounter*. http://www.cadence.com.
[2] S. R. Arikati and R. Varadarajan. A signature based approach to regularity extraction. In *Proc. of ICCAD*, pages 542–545, 1997.
[3] B. Beizer. *Software testing techniques*. Dreamtech Press, 2002.
[4] S. Brooks and B. Morgan. Optimization using simulated annealing. *The Statistician*, pages 241–257, 1995.
[5] M. Cho, H. Xiang, H. Ren, M. M. Ziegler, and R. Puri. Latchplanner: Latch placement algorithm for datapath-oriented high-performance VLSI designs. In *Proc. of ICCAD*, pages 342–348, 2013.
[6] S. Chou, M.-K. Hsu, and Y.-W. Chang. Structure-aware placement for datapath-intensive circuit designs. In *Proc. of DAC*, pages 762–767, 2012.
[7] A. Chowdhary, S. Kale, P. Saripella, N. Sehgal, and R. Gupta. A general approach for regularity extraction in datapath circuits. In *Proc. of ICCAD*, pages 332–339. IEEE, 1998.
[8] A. Chowdhary, S. Kale, P. K. Saripella, N. K. Sehgal, and R. K. Gupta. Extraction of functional regularity in datapath circuits. *IEEE Tran. on CAD*, 18(9):1279–1296, 1999.
[9] S. Das and S. P. Khatri. An efficient and regular routing methodology for datapath designs using net regularity extraction. *IEEE Tran. on CAD*, 21(1):93–101, 2002.
[10] C.-C. Huang, H.-Y. Lee, B.-Q. Lin, S.-W. Yang, C.-H. Chang, S.-T. Chen, and Y.-W. Chang. Detailed-routability-driven analytical placement for mixed-size designs with technology and region constraints. In *Proc. of ICCAD*, 2015.
[11] E. O. Hwang. Digital logic and microprocessor design. *La Sierra University, Riverside*, 2005.
[12] R. X. Nijssen and J. A. Jess. Two-dimensional datapath regularity extraction. In *IFIP Workshop on Logic and Architecture Synthesis*, pages 110–117, 1996.
[13] Rao and Kurdahi. Partitioning by regularity extraction. In *Proc. of DAC*, pages 235–238, 1992.
[14] R. Sedgewick and K. Wayne. Algorithms. pages 661–666, 2011.
[15] Synopsys. *IC Compiler User Guide: Implementation*, 2013.
[16] H. Szu and R. Hartley. Fast simulated annealing. *Physics letters A*, 122(3):157–162, 1987.
[17] S. Ward, D. Ding, and D. Z. Pan. PADE: a high-performance placer with automatic datapath extraction and evaluation through high dimensional data learning. In *Proc. of DAC*, pages 756–761, 2012.
[18] S. I. Ward, M.-C. Kim, N. Viswanathan, Z. Li, C. Alpert, E. E. Swartzlander Jr, and D. Z. Pan. Keep it straight: teaching placement how to better handle designs with datapaths. In *Proc. of ISPD*, pages 79–86. ACM, 2012.
[19] H. Xiang, M. Cho, H. Ren, M. Ziegler, and R. Puri. Network flow based datapath bit slicing. In *Proc. of ISPD*, pages 139–146, 2013.