VLSI Design

New Algorithmic Techniques for Complex EDA Problems

Guest Editors: Shantanu Dutt, Dinesh Mehta, and Gi-Joon Nam

New Algorithmic Techniques for Complex EDA Problems

VLSI Design

New Algorithmic Techniques for Complex EDA Problems

Guest Editors: Shantanu Dutt, Dinesh Mehta,
and Gi-Joon Nam

Copyright © 2014 Hindawi Publishing Corporation. All rights reserved.

This is a special issue published in "VLSI Design." All articles are open access articles distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Editorial Board

Roc Berenguer, Spain
Chien-In Henry Chen, USA
Kiyoung Choi, Korea
Anh Tuan Do, Singapore
Ethan Farquhar, USA
Dimitri Galayko, France
David Hernandez-Garduno, USA
Lazhar Khriji, Oman
Israel Koren, USA
David S. Kung, USA
Chang-Ho Lee, USA
Marcelo Lubaszewski, Brazil
Mohamed Masmoudi, Tunisia
Antonio Mondragon-Torres, USA
Jose Carlos Monteiro, Portugal
Fateme Moradi, Iran
Farshad Moradi, Denmark
Maurizio Palesi, Italy
Rubin A. Parekhji, India
Zebo Peng, Sweden
Gregory Peterson, USA
A. Postula, Australia
M. Renovell, France
Peter Schwarz, Germany
Jose Silva-Martinez, USA
Luis Miguel Silveira, Portugal
Antonio G. M. Strollo, Italy
Junqing Sun, USA
Rached Tourki, Tunisia
Spyros Tragoudas, USA
Sungjoo Yoo, Korea
Avi Ziv, Israel

Hindawi Publishing Corporation
VLSI Design
Volume 2014, Article ID 134946, 2 pages
http://dx.doi.org/10.1155/2014/134946

Editorial
New Algorithmic Techniques for Complex EDA Problems

Shantanu Dutt,[1] Dinesh Mehta,[2] and Gi-Joon Nam[3]

[1] *Department of Electrical and Computer Engineering, University of Illinois at Chicago, Chicago, IL 60607, USA*
[2] *Department of Electrical Engineering and Computer Science, Colorado School of Mines, Golden, CO 80401, USA*
[3] *IBM Research Labs, Austin, TX 78758, USA*

Correspondence should be addressed to Shantanu Dutt; dutt@ece.uic.edu

Received 3 February 2014; Accepted 3 February 2014; Published 27 April 2014

Copyright © 2014 Shantanu Dutt et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

VLSI design is a very complex problem in today's decananometer technology and billion-plus gate designs. There are two broad sets of effects that result from the rapidly decreasing feature sizes in CMOS VLSI: (a) significant increase in the number and the diversity of systems that are implemented on a single chip and (b) further exacerbation of old problems and the introduction of new ones, such as high power dissipation and temperature hot spots at all levels of the design process, high process-voltage-temperature based variability in various CMOS and interconnect parameters, and reliability of the design stemming from reduced feature sizes, to name a few. These issues present significant challenges to the entire range of EDA tools from system-level to gate-level synthesis to more accurate and computationally tractable analysis methods to better reliability via robust design and effective testing techniques. However, current algorithmic approaches used to tackle these issues lack the necessary optimization and analysis heft needed, while also being computationally tractable. Thus we feel that significant algorithmic innovations are needed to tackle these new problems with efficiency and efficacy at various stages of the VLSI design flow.

Included in this special issue are six papers that present novel algorithms for a number of the aforementioned issues. A high-level meta-algorithm for scheduling a chain of coarse grained tasks on a linear array of FPGAs, which can tackle a wide range of problem formulations and cost functions, is presented in the first paper "*Meta-algorithms for scheduling a chain of coarse-grained tasks on an array of reconfigurable FPGAs*" by D. P. Mehta, C. Shetters and D.W. Bouldin. The second paper "*Power-driven global routing for multi-supply voltage domains*" by T.-H. Wu, A. Davoodi, and J. T. Linderoth, presents a new formulation for the power-aware routing problem for multi-supply voltage designs. The problem is solved using integer programming and parallel processing, the latter being an aspect that is actively being explored for various time-intensive EDA problems. In the third paper "*Fast and near-optimal timing-driven cell sizing under cell area and leakage power constraints using a simplified discrete network flow algorithm*," H. Ren and S. Dutt develop a simplified and fast discretized network flow (DNF) algorithm for cell sizing for timing optimization under power and area constraints. This technique is much faster than a "full-fledged" DNF algorithm, but provides close to optimal solutions. The fourth paper "*A graph-based approach to optimal scan chain stitching using RTL design descriptions*" by L. Zaourar, Y. Kieffer, and C. Aktouf, presents a first-time mathematical formulation of the problem of scan insertion at the register transfer level of a design, and solves it as a traveling salesman problem. The work presented in the fifth paper "*A novel framework for applying multiobjective GA and PSO based approaches for simultaneous area, delay, and power optimization in high level synthesis of datapaths*" by D. S. Harish Ram, M. C. Bhuvaneswari and S. S. Prabhu, provides an application of the non-dominated sorting genetic algorithm (NSGA II) to the problem of multi-objective optimization in high-level synthesis with the goal of achieving solutions that are close to the true Pareto front of optimal solutions. Finally, the sixth paper "*Line search-based inverse lithography technique for mask design*" by X. Zhao and C. Chu, presents

a novel enhanced mask design method via a highly efficient gradient-basedsearch technique that results in fewer pattern errors than previous work;it thus also reduces variability.

We hope the readers will find the papers interesting and informative, and that this special issue will get researchers thinking about developing new algorithmic approaches to effectively solving critical problems in current VLSI design.

Shantanu Dutt
Dinesh Mehta
Gi-Joon Nam

Hindawi Publishing Corporation
VLSI Design
Volume 2013, Article ID 249592, 13 pages
http://dx.doi.org/10.1155/2013/249592

Research Article

Meta-Algorithms for Scheduling a Chain of Coarse-Grained Tasks on an Array of Reconfigurable FPGAs

Dinesh P. Mehta,[1] Carl Shetters,[2] and Donald W. Bouldin[3]

[1] *Department of Electrical Engineering and Computer Science, Colorado School of Mines, Golden, CO 80401, USA*
[2] *Aerospace Testing Alliance (ATA), Arnold Air Force Base, TN 37389, USA*
[3] *Department of Electrical and Computer Science, University of Tennessee, Knoxville, TN 37996, USA*

Correspondence should be addressed to Dinesh P. Mehta; dmehta@mines.edu

Received 30 April 2012; Revised 31 August 2013; Accepted 9 November 2013

Academic Editor: Shantanu Dutt

Copyright © 2013 Dinesh P. Mehta et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

This paper considers the problem of scheduling a chain of n coarse-grained tasks on a linear array of k reconfigurable FPGAs with the objective of primarily minimizing reconfiguration time. A high-level meta-algorithm along with two detailed meta-algorithms (GPRM and SPRM) that support a wide range of problem formulations and cost functions is presented. GPRM, the more general of the two schemes, reduces the problem to computing a shortest path in a DAG; SPRM, the less general scheme, employs dynamic programming. Both meta algorithms are linear in n and compute optimal solutions. GPRM can be exponential in k but is nevertheless practical because k is typically a small constant. The deterministic quality of this meta algorithm and the guarantee of optimal solutions for all of the formulations discussed make this approach a powerful alternative to other metatechniques such as simulated annealing and genetic algorithms.

1. Introduction

In this paper, we consider the problem of scheduling a chain of coarse-grained tasks on a linear array of reconfigurable FPGAs. This scheduling problem arose in the development of the CHAMPION software environment for mapping image processing applications onto an adaptive computing system (ACS). CHAMPION [1], which was developed at the University of Tennessee, maps a high-level data-flow diagram developed using the Cantata graphical programming software [2, 3] onto an ACS. CHAMPION maps the workspace to a netlist and performs data width matching and synchronization, followed by partitioning. Each resulting subnet-list is translated into structural VHDL which is merged with VHDL files specifying the ACS architecture and synthesized using commercial synthesis tools. The partitioning step which is described in detail in [4] takes as input a directed acyclic graph (DAG) and computes a partition $P = \{P_1, P_2, \ldots, P_n\}$ such that each P_i can be implemented on a single target FPGA. The partition algorithm must consider (1) the capacity of the partition (2) the number of I/O pins, and (3) the limit on RAM access modules. If n is greater than the number of FPGAs on board (k), multiple configurations of the FPGA (i.e., temporal partitioning) will be necessary. To reduce design complexity, a design decision was made to use a constrained configuration of the boards such that all signals pass in one direction along the array of FPGAs. The signal flow must accordingly be unidirectional from P_1 to P_n on the task chain. The input DAG can be linearized easily using topological sorting. However, the partitioning step must maintain the unidirectional signal flow when it allocates multiple DAG nodes to individual tasks on the task chain. The DARPA image processing application was successfully mapped onto the Wildforce-XL, the SLAAC, and MSP FPGA boards. Examples of commercial boards with multiple state-of-the-art Xilinx and Altera FPGAs can be found at http://www.dinigroup.com/new/products.php. The number of FPGAs on these boards is typically six or less although the DN7020K10 and DNDPBS327 boards have 20 and 27 FPGAs, respectively.

The result of the partitioning algorithm described above (and the input to the formulations described in this paper) is a directed chain of coarse-grained tasks such that exactly one

task fits on an FPGA. The objective of this paper is to design scheduling meta-algorithms that optimally map tasks to FPGAs under some measure of cost. Our two algorithmic schemes are general purpose reconfiguration meta-algorithm (GPRM) and the special purpose reconfiguration meta-algorithm (SPRM). Both are meta-algorithms in that they can each address several design scenarios. (The meta-algorithm concept is discussed in more detail in Section 2.)

Our focus in this paper is primarily on minimizing reconfiguration time (although it is possible to incorporate execution time into our cost formulations) because they typically dominate execution times by several orders of magnitude. For example, Natarajan et al. [1] reported that the hardware execution time to process one image using an ACS implementation of ATR, an automatic target recognition algorithm, was 33 milliseconds, as compared to nearly seven seconds needed for the entire execution. Gajjala Purna and Bhatia [5] reported that the hardware execution time for a set of four applications was on the order of tens of microseconds, while a single reconfiguration requires 242 milliseconds on the RACE architecture. More recently, Birla and Vikram [6] reported that an integral image computation was executed in 12.36 μs with reconfiguration times ranging from 3.309 to 52.944 ms (depending on the configuration clock frequency). A feature extraction and classification computation had an execution time of 8.66 μs with reconfiguration times ranging from 3.392 to 54.268 ms.

Although the original motivation for this work was to configure a board with several FPGAs, the underlying abstraction employed by our meta-algorithms can also, in principle, capture partial reconfiguration that allows "specific regions of the FPGA to be reprogrammed with new functionality while applications continue to be run in the remainder of the device." [7]. The performance of partial reconfiguration continues to be an active area of research, with recent work being focused on developing cost models that characterize reconfiguration times accurately [8–10]. Accurate cost models provide reconfiguration time data that is vital to scheduling algorithms such as SPRM and GPRM presented in this paper.

Section 3 formulates the problem and relates it to existing research. In Sections 4 and 5, respectively, we present the SPRM and GPRM meta-algorithms along with several specific instantiations. Section 6 combines the individual formulations into detailed SPRM and GPRM meta-algorithms. Section 7 contains experimental results and Section 8 concludes the paper.

2. New Algorithmic Techniques Used

A meta-algorithm is a high-level algorithmic strategy that is somewhat independent of the detailed algorithms used for solving precise instantiations of the problem. One may view techniques such as simulated annealing and genetic algorithms as meta-algorithms. These techniques have to be customized to the precise problem being solved. For example, simulated annealing may be customized by instantiating the concept of *state*, *cost*, and *move* for a given problem. Because of their stochastic nature (the use of randomness is an integral component of these techniques), simulated annealing and genetic algorithms have been widely used to solve a very large and diverse set of problems. However, the stochastic nature of these techniques leads to some difficulties in their deployment: (1) experimental results are not guaranteed to be reproducible and (2) the selection of parameters (such as the cooling factor in simulated annealing) has an appreciable impact on the runtime of the algorithm and the quality of solution. Although there are guidelines and best practices on choosing parameters, the practitioner does not know a priori what parameters will work well; considerable effort has to be expended in trial-and-error. It is also possible that different parameters are needed for different inputs.

The idea of a meta-algorithm can also refer to high-level strategies for a *specific* problem such as hierarchical web caching [11]: this paper experiments with challenging the high-level implicit strategy used in hierarchical caching that "a hit for a document at an l-level cache leads to the caching of the document in all intermediate caches (levels $l-1,\ldots,1$) on the path towards the leaf cache that received the initial request." These alternative strategies for being more selective in choosing the caches that store a local copy of the requested document are considered to be meta-algorithms because they operate independently of the actual replacement algorithm running in each individual cache.

Another scenario where meta-algorithms can be useful is when there are several detailed algorithms available to solve different variations of a problem. In this case, the meta-algorithm provides a procedure that navigates the available options and chooses which detailed algorithm to deploy. A simple example of this occurs when a difficult problem might be solved using an exhaustive algorithm for small input sizes (e.g., $n < 20$) and a heuristic or an approximation algorithm for large input sizes (e.g., $n \geq 20$). In this case, the (simple) meta-algorithm chooses which detailed algorithm to deploy. If a better heuristic or approximation algorithm is discovered for $n \geq 20$, the original one can be replaced without impacting the meta-algorithm. This approach has been used to assist a user to automatically select most suited algorithms during data mining model building process [12].

To the best of our knowledge, the use of meta-algorithms in VLSI design automation (and in configuration minimization problems, in particular) is novel.

There are two respects in which we describe meta-algorithms for this problem.

(1) This paper describes two algorithms (SPRM and GPRM) to minimize reconfiguration time of a sequence of coarse-grained tasks that are to be executed in a linear array of FPGAs. SPRM is based on dynamic programming while GPRM is based on a shortest-paths formulation. SPRM is faster but is concerned with a special case of the problem (where cost is a function of adjacent tasks in the chain), while GPRM is concerned with the general case (where cost is a function of all tasks in adjacent configurations). The SPRM and GPRM methods are themselves meta-algorithms that can be customized to address several

```
INPUT: Task Model T, Implementation Model I, Cost Model C
// I refers to the configuration of the underlying FPGA hardware
(1) if (T is a Dag and I is unstructured)
(2)    then Use optimal LIU-based algorithm [13]
(3) else if (T is a Chain or a Loop and I is a single unit/unstructured)
(4)    then Use Dynamic Programming [14]
(5) else if (T is a Chain and I is a Linear Array)
(6)    then if (C is a function of adjacent tasks)
(7)       Use SPRM
(8)          else // C is a function of all tasks in a configuration
(9)             Use GPRM.
end
```

Algorithm 1: Metaheuristic that provides a high-level framework for choosing an appropriate algorithm based on the task, implementation, and cost models being considered.

detailed problem formulations and cost specifications. These formulations include reconfiguration time minimization, dynamic task generation, repeated and similar tasks, implementation libraries, and limited cycles, all of which are discussed later.

(2) Along with Ghiasi et al. [13] and Bondalapati and Prasanna [14], SPRM and GPRM represent a novel use of these techniques in the configuration of *coarse-grained tasks* with the goal of minimizing reconfiguration time on reconfigurable logic. Following the approach of Fan and Lei [12], we present a framework within which these algorithmic techniques can be deployed. We present a metaheuristic (Algorithm 1) that provides a high-level framework for choosing an appropriate algorithm based on the task, implementation, and cost models being considered.

3. Previous Work and Problem Formulation

3.1. Previous Work. Some of the key differences between the work presented in this paper and early dynamic reconfiguration algorithms [5, 15–19] are that (1) our tasks are more coarse-grained, whereas the previous algorithms operate at the netlist level. (2) We assume that an array of FPGAs is available, whereas the previous work assumes that there is a single dynamically reconfigurable FPGA. (3) Finally, we assume that the application represented by a directed acyclic graph (DAG) of tasks has been converted into a linear chain of tasks in a precomputation step. The mapping of a DAG to a linear chain and the rationale and implications of working with linear chains are discussed in the next subsection.

In addition, there are two relatively recent papers that consider coarse-grained tasks. However, neither of these allows for a linear array of FPGAs. We describe their work in greater detail below.

Bondalapati and Prasanna [14] consider a formulation where a chain of tasks is implemented on a single configurable logic unit. A single task is executed on the configurable logic unit at a time. When its execution is complete, the logic unit is reconfigured to execute the next task in the list and so on. Each task can be implemented using multiple configurations. The choice of configurations for consecutive tasks affects the reconfiguration time. A dynamic programming algorithm optimizes the total reconfiguration time in $O(nc^2)$ time, where n is the number of tasks and c is the number of configurations for each task. Our formulation fundamentally differs from theirs in that our chain of tasks will be implemented on a *linear pipelined array* of k configurable logic units. Indeed, their dynamic programming solution may be viewed as a special case of our shortest-paths solution to our GPRM formulation when $k = 1$. (We note that Bondalapati and Prasanna further consider the case where their chain of tasks is contained in a loop requiring repeated execution and unrolling, which is outside the scope of our work.)

Ghiasi et al. [13] consider a formulation where a directed acyclic graph (DAG) of tasks has to be scheduled and executed on partially reconfigurable hardware with capacity k; that is, at most k tasks may be allocated at a time. Different tasks in the DAG may be identical, so savings in reconfiguration time may be obtained by caching a task in the partially reconfigurable hardware (PRH). Their algorithms are based on extending solutions to paging problems. This is possible because, in their formulation, *any task can be placed in any of the k locations in the PRH*, which gives them much more flexibility. Our tasks must be placed on a linear pipelined array in the order in which they appear in the chain to facilitate signal flow. Further, they do not address some of the formulations addressed in the paper (such as dynamic task generation and implementation libraries).

3.2. Discussion of Design Context. In this section we discuss the rationale and implications for the following design decisions.

(1) *Model of computation*: the DAG has traditionally been used as a general model of computation in the literature. As described earlier, our approach is to preprocess the DAG, transforming it into a linear chain of tasks and then applying our algorithms to the task chain. In principle, this preprocessing can be achieved by performing a topological sort on the DAG. Edges connecting vertices that are not adjacent in the chain

can be accommodated by including simple tasks in the intermediate chain vertices that facilitate the passing-through of the data. The existence of many such edges will add overhead, possibly making this approach impractical. Other considerations during this transformation are discussed in [4].

(2) *Implementation model*: FPGA boards contain a routing switch which permits a portion of the I/O of any FPGA to be interconnected to a portion of the I/O of any other FPGA on the board. In this work, we are using a constrained configuration of the boards such that signals flow in one direction along the array of FPGAs.

The rationale for these decisions is that many applications (e.g., image processing) can be naturally decomposed into a linear sequence of steps which map directly into a chain. For these applications, the linear flow of data results in communication paths between all of the FPGAs that are identical and predictable. This greatly simplifies system design and makes it more amenable to automation.

Some of the implications and consequences of these decisions are listed below.

(1) The traditional advantage of using a DAG computation model over a linear chain is that it permits tasks to be executed in parallel on different processors. In current FPGA systems, the response time is dominated by FPGA reconfiguration delays and not computation time [1, 5, 6]. This, along with the limited number of FPGAs available on a board, minimizes the traditional advantage of working directly with a DAG. Note, however, that our approach does not preclude the use of fine-grained parallelism in the implementation of a task. In other words, the implementation of that task in an FPGA could require several control logic blocks (CLBs) to execute in parallel. In the event that computation time is significant, our approach permits reconfigurating earlier FPGAs in parallel with ongoing computations in later FPGAs [20] and also permits reconfiguring FPGAs in parallel.

(2) Since FPGA technology results in fast execution times for complex algorithms, it is important to ensure that the scheduling algorithm itself does not consume an excessive amount of time. An advantage of linear chains is that scheduling algorithms are significantly faster than those for DAGs.

(3) A linear chain implementation of an algorithm makes it more amenable to pipelining. Thus, several images can be pipelined through an FPGA board configuration and the partial results stored. The board can then be reconfigured and the partial results pipelined through a new board configuration. This amortizes reconfiguration time over several images.

3.3. Consolidated Problem Formulation. As mentioned previously, the meta-algorithms described here capture several scenarios. Each such scenario will require a subtly different problem formulation. We begin by presenting two abstracted formulations that capture the essence of the problems solved by SPRM and GPRM: we describe (1) input, (2) output, and (3) an abstracted cost function.

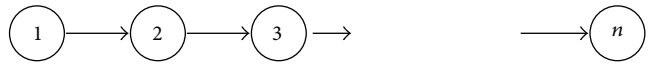


Figure 1: Input task chain.

Table 1: Two-dimensional array R representing the assignment of tasks to FPGAs and board configurations. Each row represents a board configuration and each column represents an FPGA. (Thus, Task 5 is assigned to FPGA 3 in Configuration 2.) Note that tasks are listed in row major order but are not necessarily consecutive. Note also that t the number of temporal board configurations is not known a priori.

Board config no.	FPGAs on board			
	1	2	3	4
1	**1**	**2**	**3**	
2		**4**	**5**	**6**
3	**7**	**8**	**9**	**10**
⋮				
⋮				
t			$n-1$	n

Input is a chain of tasks labeled 1 through n as shown in Figure 1 and an integer k denoting the number of FPGAs on a board.

Output is an optimal assignment of each task i to a temporal board configuration and an FPGA denoted by (B_i, F_i). B_i is an integer between 1 and t (the total number of board configurations) that denotes the temporal board configuration that Task i is assigned to. F_i is an integer between 1 and k (the number of FPGAs on the board) that denotes the FPGA that Task i is assigned to. The output may be visualized as the two-dimensional array R shown in Algorithm 1. In this example, $B_5 = 2$ and $F_5 = 3$ because Task 5 is assigned to FPGA 3 in Board Configuration 2. Note that some entries may be left blank (e.g., (1, 4) and (2, 1) in Table 1) in the optimal solution found by our algorithms. As we will see later, this could happen as a result of scheduling repeated tasks on the same FPGA in consecutive board configurations in order to minimize reconfiguration time.

Linear Task Order Constraint. Any pair of tasks i and j with $i < j$ must satisfy the following constraints: either $B_i < B_j$ (i.e., task i appears in an earlier board configuration than task j) or $B_i = B_j$ and $F_i < F_j$ (i.e., tasks i and j appear in the same board configuration but i is assigned to an FPGA that appears earlier in the pipeline than the FPGA allocated to j).

We have not yet defined optimality. We consider two cost scenarios that lead to a fundamental distinction between the SPRM and GPRM strategies.

(1) The cost to reconfigure from board Configuration c to board Configuration $c+1$ is purely a function f of the last task denoted L_c in Configuration c (highest numbered task appearing in Row c of the output table) and the first task in Configuration $c+1$ (lowest numbered task appearing in Row $c+1$ of the output table), which is $L_c + 1$. In the example, the cost to reconfigure from Configuration 1 to Configuration 2 would be $f(3, 4)$ because Task 3 is the last task in Configuration 1 and Task 4 is the first task in Configuration 2. In this restricted model, the other tasks allocated to the configurations such as Task 2 in Configuration 1 and Task 5 in Configuration 2 are assumed to have no bearing on the cost. This scenario arises when reconfiguration time is dominated by the time needed to store intermediate results between two adjacent configurations. The total cost is formally defined as $\sum_{c=1}^{t-1} f(L_c, L_c+1)$. The SPRM scheme computes an optimal solution for this scenario.

(2) Cost to reconfigure from Configuration c to Configuration $c+1$ is a function g of all of the tasks assigned to Configurations c and $c+1$. This captures the scenario when CLB reconfiguration dominates reconfiguration time. In this scenario, repeated tasks placed on the same FPGA in consecutive configurations result in significant savings.
The total cost is formally defined as $\sum_{c=1}^{t-1} g(R[c, \cdot], R[c+1, \cdot])$, where $R[c, \cdot]$ denotes all tasks in Configuration c (i.e., Row c in the output table R). Note that this scenario subsumes the SPRM scenario. The GPRM scheme is used to compute an optimal solution for this more general scenario.

4. SPRM

We present three formulations of the SPRM problem. The first is a basic formulation, the second permits dynamic task generation, and the third permits limited use of bidirectional edges and cycles. The three scenarios are quite different, but all of them share the property that the problem is fully specified by specifying costs between adjacent tasks in the chain.

4.1. SPRM Formulation 1: Basic. Given a chain of n tasks and a cost $C_{i+1} = f(i, i+1)$ of separating tasks i and $i+1$ into different configurations, compute a set of cuts of minimum total cost such that each configuration (represented by the tasks between adjacent cuts) has k or less tasks.

Application. In a chain of tasks, data is propagated along the pipeline from task i to the next task $i+1$. When these tasks belong to FPGAs in the same board configuration, the transfer can take place directly. However, when they are *not* in the same board configuration, the data must be stored in memory by task i and read from memory by task $i+1$ after the board is reconfigured so that task $i+1$ may proceed with its computation. C_{i+1} denotes this cost and could be substantial in case the data consists of images or video.

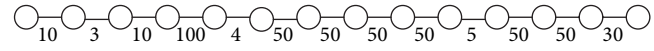


Figure 2: Initial task list with reconfiguration costs.

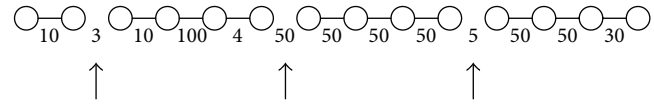


Figure 3: Task list with optimum cuts.

Consider the following simple example with $n = 14$ and $k = 4$. The costs C_i associated with each pair of consecutive tasks are shown in Figure 2.

The optimal solution consists of making cuts at edges 2, 6, and 10 with costs 3, 50, and 5, respectively, resulting in a cost of 58. The cuts are shown in Figure 3.

The dynamic programming algorithm is described below. We employ two one-dimensional arrays *cost* and *firstCut* of size n (where n is the number of tasks in the chain). The element $cost[l]$ will, upon completion of the algorithm, contain the cost of an optimal set of cuts separating the tasks in the subchain from l to $n-1$. Note that this value will be zero if $n - l \leq k$, since no cuts are required to separate a chain consisting of k or less tasks. The element $firstCut[l]$ will contain the location of the first cut in the subchain from l to $n-1$ in an optimal solution. The algorithm is outlined in Algorithm 2.

The elements of the *cost* and *firstCut* arrays are computed in reverse order. Lines 3 and 4 consider the case where the task chain from l to $n-1$ contains k or less tasks, whereas lines 5–9 consider the general case. The general case is addressed by considering all the ways in which the first cut in the subchain starting at l can be made. For each possibility, we use the value of cost $[l+i+1]$ that was computed in a previous iteration and C_{l+i+1}.

The *cost* array upon completion of execution of the algorithm on the input of Figure 2 is shown in Table 2 and the *firstCut* array in Table 3.

Theorem 1. *Algorithm COST (Algorithm 2) computes an optimal solution to SPRM Formulation 1.*

Proof. Clearly, the algorithm returns the correct value (zero) when $n - l \leq k$, since all the tasks can be accommodated in one configuration and there is no need to incur reconfiguration costs. If $n-l > k$, it must be the case that at least one cut is needed. Our dynamic programming solution considers all k possibilities for the leftmost cut and uses previously computed optimal cost values for the subchains. □

Theorem 2. *Algorithm COST has time complexity* $O(n \log k)$ *and space complexity* $O(n)$.

Proof. There are $O(n)$ elements in the *cost* array, each of which is computed in $O(k)$ time. This results in a complexity of $O(nk)$. We note that this can be improved to $O(n \log k)$ as follows: observe that in order to compute any element $cost[l]$, we obtain the minimum from a set of k quantities of the form

```
COST(n)
{Compute minimum cost using dynamic programming}
(1) for l = n − 1 down to 0 do // n is number of tasks in chain
(2)    cost[l] = ∞
(3)    if (n − l) ≤ k then //Base case
(4)       cost[l] = 0
(5)       firstCut[l] = n
(6)    else // General case
(7)       for i in [0, k − 1] do // Try k cuts, determine which is best
(8)          tempCost = cost[l + i + 1] + C_{l+i+1}
(9)          if tempCost < cost[l] then
(10)            cost[l] = tempCost
(11)            firstCut[l] = l + i + 1
end
```

Algorithm 2: SPRM (Straight-Cut) algorithm.

Table 2: Contents of array *cost* on completion of algorithm.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
58	58	55	55	55	55	5	5	5	5	0	0	0	0

Table 3: Contents of array *first cut* on completion of algorithm.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
2	2	6	6	6	6	10	10	10	10	14	14	14	14

```
CUTS(n) // The traceback step
(1) next = 0
(2) while next < n do
(3)    output firstCut[next]
(4)    next = firstCut[next]
end
```

Algorithm 3: Output-Cuts algorithm.

$cost[l+i+1]+C_{l+i+1}$. Also, observe that in the computation of $cost[l-1]$, $k-1$ of these quantities are identical to those used for $cost[k]$. Thus, it is possible to use a min-heap data structure in combination with a queue of pointers to elements in the min-heap to (1) compute the minimum, (2) delete one element, and (3) insert one element per iteration. Each of these can be accomplished in $O(\log k)$ time, giving the result. □

The algorithm to output cuts is given in Algorithm 3.

4.2. SPRM Formulation 2: Dynamic Task Generation. The dynamic task generation problem is a modification of the formulation described above. As before, the input is a linear chain of tasks. However, in addition to the C_{i+1} costs associated with each edge, a *boolean parameter is also associated with each node*. If this parameter is set to true a cut on either side of this node, this will require the creation of an additional node. This additional node will occupy one of the available FPGAs and therefore directly affects the calculation of the optimum cost and cut-set in the algorithm.

Application. When a task is placed in the first FPGA in a configuration, it will need to read intermediate data from memory. Similarly, when a task is placed in the last FPGA in a configuration, it will need to write intermediate data to memory. This additional read/write functionality must be implemented on the FPGA. However, if the additional logic required does not fit on the FPGA or if the existing functionality on the FPGA already contains memory accesses and more memory accesses are not permitted by the architecture, an additional task must be created and accommodated on another FPGA.

We use the same example as before, except that along with the reconfiguration cost there is a boolean flag associated with each vertex (Figure 4).

Our algorithm obtains cuts at 3, 5, 6, and 10 of costs 10, 4, 50, and 5 resulting in an optimal cost of 69. Note that the addition of extra task nodes affects the cut location and therefore the overall cost. The optimum cuts would break the task list as can be seen in Figure 5.

The cost function, which is the main modification to the Straight-Cut algorithm, is displayed in Algorithm 4.

The main difference between the dynamic task and the straight cut versions is that configurations are analyzed to determine how many of their end-nodes are marked (i.e., an additional node is required if the cut is at this location). This quantity (0, 1, or 2) is subtracted from the possible number of FPGAs available for a configuration.

```
COST(n)
{Compute minimum cost using dynamic programming}
(1) for l = n − 1 down to 0 do // n is number of tasks in chain
(2)    cost[l] = ∞
(3)    if (n − l) ≤ k − b[l] − b[n − 1] then //Base case
(4)       cost[l] = 0
(5)       firstCut[l] = n
(6)    else // General case
(7)       for i in [0, k − 1 − b[l]] do // try all cuts, determine best one
(8)          if i < k − 1 − b[l] or b[l + i] = 0 then // checks whether additional task needed
(9)             tempCost = cost[l + i + 1] + C_{l+i+1}
(10)            if tempCost < cost[l] then
(11)               cost[l] = tempCost
(12)               firstCut[l] = l + i + 1
end
```

Algorithm 4: SPRM (dynamic-node generation) algorithm.

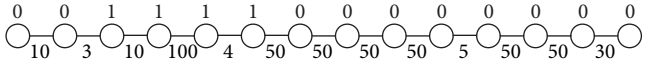


Figure 4: Initial task list with reconfiguration costs and boolean flag.

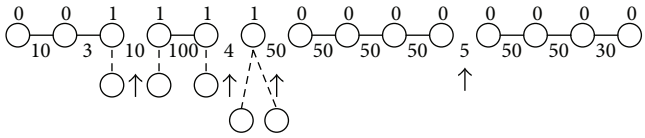


Figure 5: Task list with optimum cuts and additional node locations.

4.3. SPRM Formulation 3: Linear Chain with Limited Cycles. Recall that the partitioning algorithm of [4] partitions a DAG into a chain of n tasks such that the signal flows from left to right. However, this may not be possible or advantageous in some applications and may result in bidirectional edges or cycles as shown in Figure 6.

The interconnect on FPGA boards can be configured to permit bidirectional or cyclic signal flow, so the only limitation is that all tasks involved in this type of signal flow must be placed in the same board configuration. (Otherwise, a task in an earlier board configuration will need data produced by a task in the future.) Although SPRM is designed for a linear chain with a unidirectional signal flow, it can be modified for situations with a *limited* number of bidirectional edges or cycles. This constraint can be accommodated by assigning a large cost M to each bidirectional edge and to all edges in a cycle as shown in Figure 7. This will discourage our dynamic programming algorithm from cutting these edges forcing the algorithm to place the relevant tasks on the same board, provided this is feasible.

5. GPRM

Recall that GPRM is designed for situations where the cost of reconfiguring Configuration c into Configuration $c + 1$ is a function of all tasks assigned to those two configurations whereas SPRM is designed for situations where the reconfiguration cost is a function of the last task assigned to c and the first task assigned to $c+1$, while ignoring all of the other tasks in the two configurations. Thus GPRM subsumes SPRM and can handle all of the cases discussed in the previous section. In this section, we present six formulations of the GPRM problem. The first three consider scenarios where reconfiguration time can be saved by reusing logic through partial reconfiguration. The fourth and fifth consider dynamic task generation and linear chains with limited cycles (which can be handled by SPRM Formulations 2 and 3, resp.), while the sixth describes how cost metrics can be extended to include execution time.

5.1. GPRM Formulation 1: Repeating Tasks. To illustrate the GPRM strategy, we consider an example consisting of a board with $k = 5$ FPGAs and a task chain with $n = 9$ tasks, some of which are repeated. (Tasks can repeat if the same image processing transformation or the same memory management step is used several times in the computation.) The task chain consists of the tasks A, B, C, C, A, B, D, E, and C. Repeating tasks are represented by using the same letter of the alphabet. Table 4 shows four possible configuration sequences. Within each of the four sequences, a row represents a configuration and indicates which task is placed in each FPGA in that configuration.

To further simplify the presentation, we assume that the cost of reconfiguring an FPGA is 1 or 0 depending on the tasks that are allocated to the FPGA in consecutive configurations. The cost is 0 in the following two cases. (1) The FPGA remains empty in consecutive configurations. (2) The FPGA contains the same task in consecutive configurations. The cost is 1 in the following three cases. (1) The FPGA is empty in one configuration and occupied by a task in the next configuration. (2) The FPGA is occupied in one configuration and empty in the next configuration (reconfiguration is needed to prevent unwanted side effects). (3) The FPGA contains different tasks

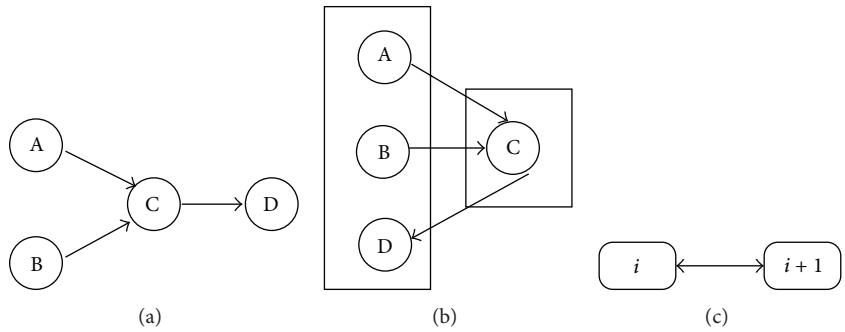


Figure 6: Allocating multiple DAG nodes to a task can lead to a bidirectional edge: (a) original DAG, (b) allocation of DAG nodes to FPGA tasks, (c) bidirectional edge resulting from node merging.

Table 4: Four possible solutions to GPRM sample problem: The first row shows the initially empty configurations. The second row shows the FPGAs with their initially loaded tasks (with the cost of loading included in the cost column in Row 2). We describe the example in detail for Solution 1: The cost of Row 2 is 3 because 3 of the 5 empty FPGAs are loaded with tasks while the other 2 remain empty. The cost of reconfiguring Row 2 into Row 3 is 4 (four FPGAs go from loaded to empty or vice versa while FPGA 3 retains the same task C). The cost of reconfiguring Row 3 into Row 4 is 4 (again four FPGAs either go from loaded to empty or vice versa while FPGA 3 continues to retain task C). We ignore the cost of emptying out the FPGAs at the end, giving a total cost of 3 + 4 + 4 = 11. Solutions 3 and 4 are the best, each with a total cost of 6.

Solution 1						Solution 2						Solution 3						Solution 4					
1	2	3	4	5	$	1	2	3	4	5	$	1	2	3	4	5	$	1	2	3	4	5	$
					—						—						—						—
A	B	C			3	A	B	C	C		4	A	B	C		C	4	A	B		C	C	4
		C	A	B	4	A	B	D	E	C	3	A	B	D	E	C	2	A	B	D	E	C	2
D	E	C			4						—						—						—
Total cost					11	Total cost					7	Total cost					6	Total cost					6

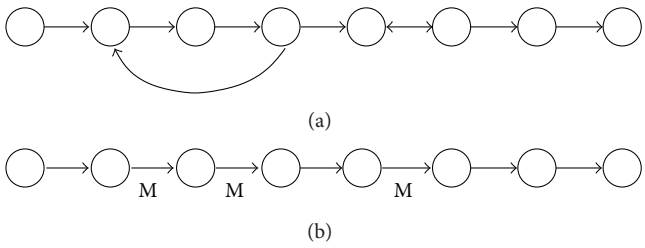


Figure 7: (a) task chain with one cycle and one bidirectional edge. (b) Cost M assigned to each bidirectional or cycle edge.

in the two consecutive configurations. Note that the reconfiguration cost is a *function of ALL tasks in consecutive configurations*, which is precisely when the GPRM strategy is applicable. Note also that the SPRM strategy would not apply in this scenario.

Our GPRM strategy consists of modeling the problem as a shortest-path problem in a directed acyclic graph with nonnegative costs assigned to the edges of the dag. Each node in the graph represents a possible configuration, that is, an assignment of tasks to FPGAs. Nodes of the graph are denoted by $[l, b_1 b_2 \cdots b_k]$, where l denotes the index of the first task included in the configuration and b_i, $1 \le i \le k$, is a bit which is set to 1 if FPGA i is occupied by a task and 0 if it is left unoccupied. Thus, we refer to $b_1 b_2 \cdots b_k$ as the *occupancy bit string*. The total number of tasks included in a configuration is $\sum_{i=1}^{k} b_i$ and the index m of the last task in the configuration is $l + \sum_{i=1}^{k} b_i - 1$. Note that empty configurations ($\sum_{i=1}^{k} b_i = 0$) or configurations where $m > n$ need not be considered. (Recall that n is the total number of tasks.)

Configuration node $[l, b_1 b_2 \cdots b_k]$ has an out-edge to all nodes of the form $[m+1, *]$, where m denotes the index of the last task node in $[l, b_1 b_2 \cdots b_k]$ and $*$ denotes any occupancy bit pattern that represents a valid configuration. Edges connect every possible pair of consecutive configurations. We assign a cost to each edge which denotes the cost of reconfiguring the FPGA array with the new set of tasks. Since this quantity is a function of the two configurations joined by the edge, it can be easily computed. We also include a dummy source and a dummy sink node to facilitate the shortest-path computation. The source represents an "empty" configuration and has an out-edge to all nodes of the form $[1, *]$. The cost of the out-edge is the cost of initially configuring the FPGA array and is trivially computed. All nodes representing a configuration that contains task n have an out-edge to the sink node of cost 0.

The DAG corresponding to our example is partially shown in Figure 8.

The four configuration sequences of Table 4 correspond to four different paths from source to sink in the DAG. Thus, Solution 1 contains the following sequence of intermediate nodes (using our notation for nodes) ([1, 11100], [4, 00111],

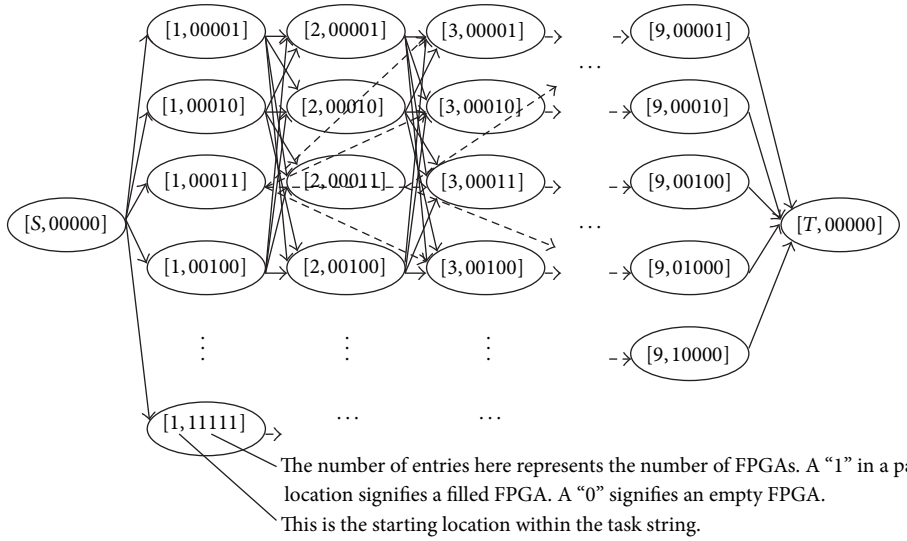


Figure 8: DAG example.

GPRM
(1) Create Dag
(2) Compute Reconfiguration Cost for each Dag edge
(3) Compute Least Cost Source-Sink Path in Dag
end

Algorithm 5: Shortest paths in DAG algorithm.

[7, 11100]) while Solution 4 contains the sequence ([1, 11011], [5, 11111]). There are two sequences that give a minimum cost and therefore an optimum solution.

The three steps of the shortest-paths technique are summarized in Algorithm 5.

Lemma 3. *The total number of configuration nodes for a chain of n tasks on an array of k FPGAs is bounded by $n2^k$.*

Proof. The variable l can take on n values and each of the k elements of the occupancy string can take on one of two values. Multiplying these quantities gives the desired result. □

Lemma 4. *The total number of edges for a chain of n tasks on an array of k FPGAs is bounded by $O(n4^k)$.*

Proof. There are $O(n2^k)$ nodes, each of which has at most 2^k out edges, yielding the desired result. □

Theorem 5. *The shortest-paths in DAG algorithm (Algorithm5) computes an optimal cost solution to GPRM Formulation 1.*

Proof. Note that the only constraint is that the chain of tasks must be executed in order. Any path from source to sink in our dag gives a valid sequence of configurations since nodes are only joined by edges if they contain consecutive task sequences. Similarly, any valid sequence of configurations is represented by a path in our graph since each valid configuration is represented by a node and since edges are constructed for all possible pairs of consecutive configurations. The total cost for a path is the sum of the cost of the edges, which represent individual reconfiguration costs. Consequently, the shortest-path from source to sink gives an optimal-cost sequence of configurations. □

Theorem 6. *GPRM requires $O(nk4^k)$ time and $O(n2^k)$ space.*

Proof. The number of nodes in the dag is $O(n2^k)$ and the number of edges is $O(n4^k)$. Edge costs are computed in $O(k)$ time per edge since a pair of tasks needs to be examined for each of k FPGAs on the board. Thus, the total time for graph creation is $O(nk4^k)$. A topological ordering based technique can be used to compute the shortest source-sink path since the graph is a dag in $O(n4^k)$ time. Thus, the total time complexity is $O(nk4^k)$. The space required by the graph is proportional to the number of edges and is $O(n4^k)$. Next, we propose a data structure to represent the DAG that reduces the memory requirements of GPRM to $(n2^k)$. We use an $n \times 2^k$ 2-dimensional array to represent the nodes of the DAG. The element in position (i, j) corresponds to the node $[i, j_1 j_2 \cdots j_k]$, where the bit string $j_1 j_2 \cdots j_k$ is the binary representation of j. Note that it is not necessary to explicitly maintain the edges leading out of the node corresponding to element (i, j). These edges can be implicitly computed during the topological traversal of the DAG as there will be an outgoing edge to each of the 2^k nodes of the form $(i+\sum_{l=1}^{k} j_l, *)$. This reduces the storage complexity to $O(n2^k)$. □

An anonymous referee asked us to consider how we might modify GPRM when the FPGA interconnection network permits signals between any pair of FPGAs leading to the observation that the linear task order constraint assumed in Table 1

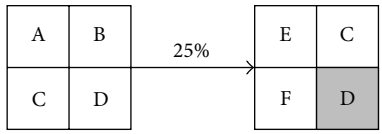


Figure 9: Similar Tasks Illustration: subtask D is repeated, giving a 25% similarity or a 0.75 partial reconfiguration cost instead of a full reconfiguration cost of 1.

need not be satisfied. While this is beyond the scope of the current paper, we briefly mention how this could be achieved. Our notation of [4, 01110] currently means that tasks 4, 5, and 6 must be assigned to FPGAs 2, 3 and 4, respectively. Under the referee's assumption any permutation of the three tasks would also be acceptable, leading to 3! = 6 configurations. The notation could be modified by replacing the "1"s in the bit string with the appropriate permutation information. Thus, [4, 01230] would mean that Tasks 4, 5, and 6 are placed in FPGAS 2, 3, and 4, respectively, as before, while [4, 02310] would mean that Tasks 4, 5, and 6 are placed in FPGAs 4, 2, and 3, respectively. In the worst case, this will expand the number of nodes in the DAG by a factor of $k!$ and the number of edges by a factor of $k!^2$.

5.2. GPRM Formulation 2: Similar Tasks. Formulation 1 assigned a reconfiguration time of 0 if the identical task was allocated to an FPGA in consecutive configurations. In Formulation 2, we consider the scenario where two tasks are not identical, but are similar, because some subtasks are identical (Figure 9). Specifically, if two tasks located in the same FPGA in consecutive configurations have identical subtasks in the same subareas of the FPGA, it may be possible to obtain the second configuration from the first by *partially* reconfiguring the FPGA. This would consume less time than a full reconfiguration and can be modeled by the DAG construction as described in the previous section by simply modifying edge costs before executing the shortest-path algorithm. Note that actual reconfiguration costs can be modeled fairly accurately using approaches discussed in [8–10].

5.3. GPRM Formulation 3: Implementation Libraries. A task can be implemented in several ways in an FPGA depending on how its subtasks are laid out within the FPGA. This gives rise to several implementations of a task that can be stored in a library. An intelligent scheduler would choose task implementations from the library that minimize reconfiguration cost. We illustrate these ideas in Figure 10. Suppose that an FPGA originally contains a task consisting of subtasks {A, B, C, D}. Suppose this FPGA is to be reconfigured with a task consisting of subtasks {C, D, E, F}. The figure shows three implementations of this task. Clearly, the choice of an implementation affects the reconfiguration cost. For example, implementation 1 results in a 0% match (and therefore a complete reconfiguration), while implementation 2 results in a 25% match (and therefore a partial reconfiguration cost of 75%), and implementation 3 results in a partial reconfiguration cost of 50%.

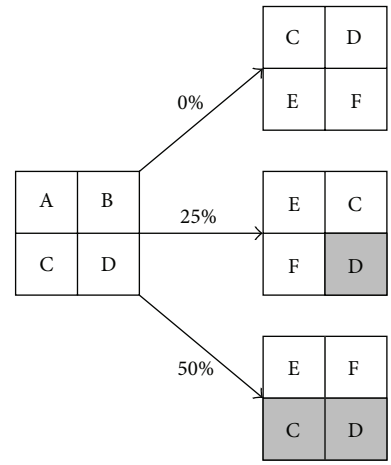


Figure 10: Implementation libraries illustration.

Assume that the maximum number of implementations of any task that are available in an implementation library is α. Then, we can construct a graph consisting of nodes of the form $[l, c_1c_2 \cdots c_k]$, where l denotes the index of the first task included in the configuration and c_i, $1 \le i \le k$, is an integer which is set to 0 if FPGA i is unoccupied or to j, $1 \le j \le \alpha$ if FPGA i is occupied by implementation number j of a task.

Configuration node $[l, c_1c_2 \cdots c_k]$ has an out-edge to all nodes of the form $[m + 1, *]$, where m denotes the index of the last task node in $[l, c_1c_2 \cdots c_k]$ and $*$ denotes any occupancy pattern that represents a valid configuration. As before, dummy source and sink nodes are added to the graph. The source has an out-edge to all nodes of the form $[1, *]$. All nodes representing a configuration that contains task n have an out-edge to the sink node of cost 0.

As before, the shortest-path from source to sink results in an optimal configuration schedule.

Lemma 7. *The total number of nodes in the graph is bounded by* $n(\alpha + 1)^k$.

Proof. The variable l can take on n values and each of the k elements of the occupancy string can take on one of $\alpha + 1$ values. Multiplying these quantities gives the desired result. □

Lemma 8. *The total number of edges for a chain of n tasks on an array of k FPGAs is bounded by* $n(\alpha + 1)^{2k}$.

5.4. GPRM Formulation 4: Dynamic Task Generation. Recall that additional functionality may be required in the first and last FPGAs to, respectively, read in and write out data in each configuration. This additional functionality may not fit in an FPGA when combined with the task assigned to that FPGA from the task chain. Recall also that this was addressed by the SPRM Formulation 2. Since GPRM is a generalization of SPRM, we expect to be able to capture this formulation within the framework of GPRM. The GPRM algorithm can be modified by simply deleting all nodes (and incident edges) in the graph that contain tasks in the first or last FPGA that cannot

accommodate the additional functionality. A shortest-path computation on the remainder of the graph deftly solves this formulation.

5.5. GPRM Formulation 5: Linear Chain with Limited Cycles. We again consider the situation described in SPRM Formulation 3 and depicted in Figure 7 (task chain with a limited number of bidirectional edges and small cycles). In SPRM, this was addressed by imposing a large cost on bidirectional edges or between adjacent tasks that were part of a cycle. This was done to prevent the scenario where a board configuration requires the results of a future configuration. The same approach can be used in GPRM: if there are a pair of consecutive configurations (represented by vertices joined by a directed edge in the shortest-path graph) such that the output of a task in the later configuration is the input of a task in an earlier configuration, the cost of the directed edge connecting the two configurations is made arbitrarily large. The shortest-path algorithm used to compute the optimal schedule will find a path that (implicitly) ignores the high-cost edge.

5.6. GPRM Formulation 6: Alternative Cost Functions. Although the preceding discussion has focused on minimizing reconfiguration time, our model can handle alternative cost formulations that (1) include execution time and (2) systems where reconfiguration of the FPGAs may be sequential or parallel. To make this concrete, we consider the two scenarios shown in Figure 11. In both scenarios, each of the four FPGA tasks is decomposed into a reconfiguration component (R) and an execution component (E). An arrow from the reconfiguration component to the execution component of each task is the dependency resulting from the observation that a task can only be executed after it is configured. In (a), the four reconfigurations are assumed to be sequential—this is denoted by the arrows from R_a to R_b to R_c to R_d. In (b) the four reconfigurations are assumed to be carried out in parallel. Thus, there are no arrows among R_a, R_b, R_c, and R_d. In both cases we assume that there is a sequential dependency between the executions (so there are arrows from E_a to E_b to E_c to E_d).

Example. Assume that reconfiguration times R_a, R_b, R_c, and R_d are 100 units each and that execution times E_a, E_b, E_c, and E_d are 5, 7, 3, and 10 units, respectively. Since execution times are smaller than reconfiguration times, the total cost of the formulation in Figure 11(a) is 400 (4 sequential reconfigurations) + 10 (execution time of E_d) = 410. The cost of the formulation in Figure 11(b) is 100 (to configure the FPGAs in parallel) + 25 (the sum of the execution times) = 125. More generally, the formal cost function that captures the dependency in Figure 11(b) is $\max(\max(\max(R_1+E_1, R_2)+E_2, R_3)+E_3, R_4)+E_4$. Substituting, we can confirm that this is evaluated to be $\max(\max(\max(100+5, 100)+7, 100)+3, 100)+10 = 125$. The formal cost function for Figure 11(a) is $\max(R_a+R_b+R_c+R_d, \max(R_a+R_b+R_c, \max(R_a+E_a, R_a+R_b)+E_b)+E_c)+E_d$, which is evaluated to be 410 as expected. Both cost functions are critical path computations that model the particular reconfiguration/execution dependencies.

```
SPRM Meta Algorithm
(1) if (Dynamic Task Generation Needed)
(2)     then Algorithm = SPRM 2
(3)     else Algorithm = SPRM 1
(4) if (Task Chain Has Cycles)
(5)     then Incorporate SPRM 3 Cost Model
end
```

Algorithm 6: Detailed SPRM Metaheuristic showing that SPRM 1 and SPRM 2 are mutually exclusive, but both can be combined with SPRM 3.

Both of these are functions of the tasks in the board configuration and fall under the framework of the GPRM model. In the GPRM shortest-path formulation, incoming edges to the vertex corresponding to this board configuration can be assigned costs computed as above. Other complex cost functions involving reconfiguration and execution times can also be similarly captured by our cost model.

6. Detailed SPRM and GPRM Meta heuristics

We have previously shown a high-level metaheuristic in Algorithm 1 that uses the cost model to determine whether to use SPRM or GPRM. We have also argued that, since the GPRM cost model subsumes the SPRM cost model, any formulation that can be addressed by SPRM can also be solved using GPRM. However SPRM is faster making it the better choice when either approach can be used.

For ease of presentation, we described three SPRM and six GPRM formulations separately in the preceding sections. However, in some cases these formulations can be combined with each other in complex ways. This is illustrated in Algorithms 6 and 7.

Finally, we note that although we have only proved optimality for SPRM Formulation 1 and GPRM Formulation 1, all of the scenarios described in the previous sections including the combinations described in Algorithms 6 and 7 can be solved optimally by using appropriate extensions of SPRM and GPRM.

7. Experimental Results

All algorithms were implemented in the C programming language on a 500 Mhz Dell OptiPlex GX1 with 256 MB of RAM. The automatic target recognition algorithm (ATR) that was implemented on CHAMPION was the only application that required board reconfigurations. Our algorithms obtained an identical (optimal) solution to that of the manual cut.

In addition to this, we designed some examples to "stress-test" our code. The results are presented below. The results for GPRM Formulation 1 (repeated tasks) with cost functions as defined in Section 5.1 are shown in Table 5. Elapsed time was reported in seconds using the clock( ) function. A naive heuristic that was used for comparison purposes simply made cuts at regular intervals of k tasks along the task chain. Note that in all cases the runtime was not more than a few seconds.

```
GPRM Meta Algorithm
(1)  if (Implementation Libraries Used)
(2)     then  DAG_Structure = Basic DAG Structure defined in GPRM 3
(3)     else DAG_Structure = Basic DAG Structure defined in GPRM 1
(4)  if (Dynamic Task Generation Needed)
(5)     Update DAG Structure as described in GPRM 4
(6)  Cost_Model = SPRM 1 Cost Model// baseline cost model
(7)  If (Repeated Tasks Present)
(8)     then Update Cost_Model as described in GPRM 1
(9)  If (Similar Tasks Present)
(10)    then Update Cost_Model as described in GPRM 2
(11)  If (Limited Cycles Present)
(12)    then Update Cost_Model as described in GPRM 5
(13)  If (Alternative Cost Functions Needed )
(14)    then Modify Cost_Model as described in GPRM 6
(15)  Run Shortest Paths Algorithm.  end
```

Algorithm 7: Detailed GPRM metaheuristic: this elaborates on the GPRM outline provided in Algorithm 5 by first determining the appropriate DAG structure. It then computes the correct cost model, which can be a combination of any of the cost models used in SPRM 1, GPRM 1, GPRM 2, GPRM 5, and GPRM 6.

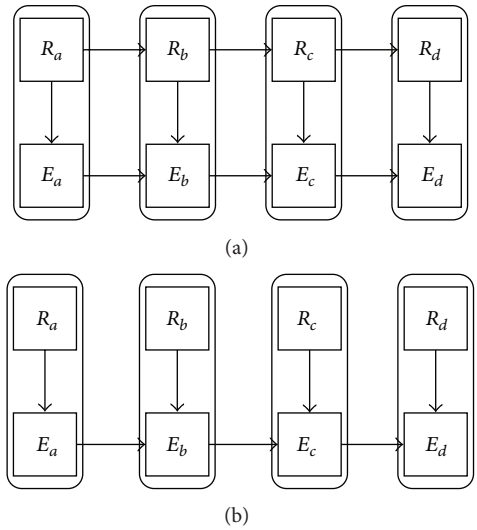


Figure 11: Both scenarios include execution time. (a) assumes sequential reconfiguration and (b) assumes parallel reconfiguration.

The runtime for the heuristic was less than the significant digits reported. The average cost reduction between the GPRM technique and the simple heuristic is 21%.

The results for SPRM Formulation 1 (the Straight-Cut version) as defined in Section 4.1 with integer costs chosen randomly in [1, 100] are shown in Table 6. Increasing the FPGA size from 4 to 6 lowered the cost substantially. The Straight Cut SPRM algorithm results in an average cost reduction of 50% relative to the naive heuristic.

The results for SPRM Formulation 2 (dynamic-node generation) as defined in Section 4.2 are shown in Table 7. We used the same data as for SPRM Formulation 1. Increasing the FPGA size from 4 to 6 did lower the cost substantially. The average cost reduction relative to the naive heuristic is 46%.

Table 5: GPRM versus heuristic.

Nodes	FPGAs	GPRM		Heuristic	
n	k	Cost	Time	Cost	Time
100	4	59	0.11	72	0.00
100	5	59	0.33	74	0.00
500	4	298	0.33	377	0.00
500	5	294	1.37	384	0.00
1000	4	600	0.82	744	0.00
1000	5	590	3.13	757	0.00

Table 6: Straight-Cut SPRM versus naive heuristic.

Nodes	FPGAs	SPRM	Heuristic
n	k	Cost	Cost
100	4	560	965
100	5	415	962
100	6	371	724
175	4	1067	1782
175	5	811	1884
175	6	683	1469

Table 7: Dynamic SPRM versus heuristic.

Nodes	FPGAs	SPRM	Heuristic
n	k	Cost	Cost
100	4	921	1810
100	5	514	1053
100	6	375	922
175	4	2082	3323
175	5	1181	2056
175	6	813	1685

The naive heuristic was modified to add task nodes depending on the values of the Boolean flag associated with each node. (If the flag is true, the task is not permitted to be placed in the first or last FPGA; instead it is placed in the next available FPGA.) (An anonymous reviewer suggested using a greedy heuristic for the SPRM problem formulations that could traverse the task chain, greedily making a cut at the smallest cost arc from among the next k arcs at each stage. We expect that this will perform much better than the naive heuristic. However, we note that this greedy approach cannot guarantee optimal costs and moreover will have essentially the same linear runtime as our dynamic programming algorithm for small constant k.)

8. Discussion and Conclusion

This paper presented a high-level meta-algorithm and detailed meta-algorithms for SPRM and GPRM for scheduling FPGA board configurations. This is a powerful strategy that encapsulates a wide range f problem formulations (including combinations of problem formulations) and algorithms into a single framework. The deterministic quality of this meta-algorithm and the guarantee of optimal solutions for all of the formulations discussed make this a viable alternative to traditional stochastic meta-algorithms such as simulated annealing and genetic algorithms.

Acknowledgments

The authors gratefully acknowledge the support of DARPA Grant no. F33615-97-C-1124 and NSF Grant no. CNS-0931748.

References

[1] S. Natarajan, B. Levine, C. Tan, D. Newport, and D. Bouldin, "Automatic mapping of Khoros-based applications to adaptive computing systems," in *Proceedings of the Military and Aerospace Applications of Programmable Devices and Technologies International Conference (MAPLD '99)*, pp. 101–107, 1999.

[2] J. R. Rasure and C. S. Williams, "An integrated data flow visual language and software development environment," *Journal of Visual Languages and Computing*, vol. 2, no. 3, pp. 217–246, 1991.

[3] J. Rasure and S. Kubica, "The khoros application development environment," Tech. Rep., Khoral Research Inc., Albuquerque, NM, USA, 2001.

[4] N. Kerkiz, A. Elchouemi, and D. Bouldin, "Multi-FPGA partitioning method based on topological levelization," *Journal of Electrical and Computer Engineering*, vol. 2010, Article ID 709487, 5 pages, 2010.

[5] K. M. Gajjala Purna and D. Bhatia, "Temporal partitioning and scheduling data flow graphs for reconfigurable computers," *IEEE Transactions on Computers*, vol. 48, no. 6, pp. 579–590, 1999.

[6] M. Birla and K. N. Vikram, "Partial run-time reconfiguration of FPGA for computer vision applications," in *Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS '08)*, April 2008.

[7] D. Dye, "Partial reconfiguration of Xilix FPGAs using ISE design suite," Tech. Rep., Xilinx, Inc., 2012.

[8] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, "Run-time partial reconfiguration speed investigation and architectural design space exploration," in *Proceedings of the 19th IEEE International Conference on Field Programmable Logic and Applications (FPL '09)*, pp. 498–502, September 2009.

[9] F. Duhem, F. Muller, and P. Lorenzini, "Reconfiguration time overhead on field programmable gate arrays: reduction and cost model," *IET Computers and Digital Techniques*, vol. 6, no. 2, pp. 105–113, 2012.

[10] K. Papadimitriou, A. Dollas, and S. Hauck, "Performance of partial reconfiguration in FPGA systems: a survey and a cost model," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 4, pp. 1–24, 2011.

[11] N. Laoutaris, S. Syntila, and I. Stavrakakis, "Meta algorithms for hierarchical web caches," in *Proceedings of the 23rd IEEE International Conference on Performance, Computing, and Communications (IPCCC '04)*, pp. 445–452, April 2004.

[12] L. Fan and M. Lei, "Reducing cognitive overload by meta-learning assisted algorithm," in *Proceedings of the IEEE International Conference on Bioinformatics and Biomedicine*, pp. 429–432, 2008.

[13] S. Ghiasi, A. Nahapetian, and M. Sarrafzadeh, "An optimal algorithm for minimizing run-time reconfiguration delay," *ACM Transactions on Embedded Computing Systems*, vol. 3, pp. 237–256, 2004.

[14] K. Bondalapati and V. K. Prasanna, "Loop pipelining and optimization for run time reconfiguration," in *Parallel and Distributed Processing*, vol. 1800 of *Lecture Notes in Computer Science*, pp. 906–915, Springer, Berlin, Germany, 2000.

[15] S. Trimberger, "Scheduling designs into a time-multiplexed FPGA," in *Proceedings of the 6th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '98)*, pp. 153–160, February 1998.

[16] H. Liu and D. F. Wong, "Network flow based circuit partitioning for time-multiplexed FPGAs," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '98)*, pp. 497–504, November 1998.

[17] H. Liu and D. F. Wong, "Circuit partitioning for dynamically reconfigurable FPGAs," in *Proceedings of the 7th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '99)*, pp. 187–194, February 1999.

[18] D. Chang and M. Marek-Sadowska, "Partitioning sequential circuits on dynamically reconfigurable FPGAs," *IEEE Transactions on Computers*, vol. 48, no. 6, pp. 565–578, 1999.

[19] D. Chang and M. Marek-Sadowska, "Buffer minimization and time-multiplexed I/O on dynamically reconfigurable FPGAs," in *Proceedings of the 5th ACM International Symposium on Field-Programmable Gate Arrays (FPGA '97)*, pp. 142–148, February 1997.

[20] S. Cadambi, J. Weener, S. C. Goldstein, H. Schmit, and D. E. Thomas, "Managing pipeline-reconfigurable FPGAs," in *Proceedings of the 6th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '98)*, pp. 55–64, February 1998.

Hindawi Publishing Corporation
VLSI Design
Volume 2013, Article ID 905493, 12 pages
http://dx.doi.org/10.1155/2013/905493

Research Article

Power-Driven Global Routing for Multisupply Voltage Domains

Tai-Hsuan Wu, Azadeh Davoodi, and Jeffrey T. Linderoth

University of Wisconsin, Madison, WI 53706, USA

Correspondence should be addressed to Azadeh Davoodi; adavoodi@wisc.edu

Received 22 June 2012; Revised 16 November 2012; Accepted 5 April 2013

Academic Editor: Shantanu Dutt

Copyright © 2013 Tai-Hsuan Wu et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

This work presents a method for global routing (GR) to minimize power associated with global nets. We consider routing in designs with multiple supply voltages. Level converters are added to nets that connect driver cells to sink cells of higher supply voltage and are modeled as additional terminals of the nets during GR. Given an initial GR solution obtained with the objective of minimizing wirelength, we propose a GR method to detour nets to further save the power of global nets. When detouring routes via this procedure, overflow is not increased, and the increase in wirelength is bounded. The power saving opportunities include (1) reducing the area capacitance of the routes by detouring from the higher metal layers to the lower ones, (2) reducing the coupling capacitance between adjacent routes by distributing the congestion, and (3) considering different power weights for each segment of a routed net with level converters (to capture its corresponding supply voltage and activity factor). We present a mathematical formulation to capture these power saving opportunities and solve it using integer programming techniques. In our simulations, we show considerable saving in a power metric for GR, without any wirelength degradation.

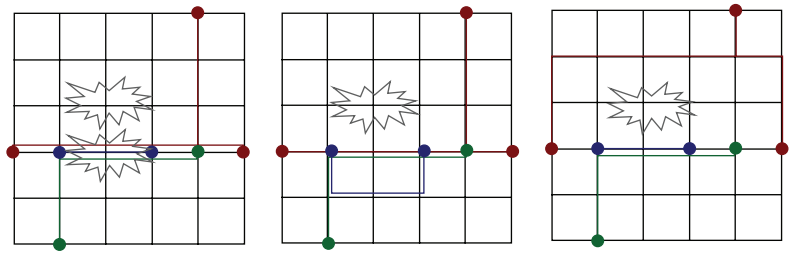
1. Introduction

Power consumption is a primary design objective in many application domains. Dynamic power still remains the dominant portion of the overall power spectrum. Design with multisupply voltage (MSV) allows significant reduction in dynamic power by taking advantage of its quadratic dependence on the supply voltage.

Dynamic power is dissipated in combinational and sequential logic cells, clock network, and the (remaining) local and global nets. We refer to the latter as the signal power. The signal power can take a significant portion of the dynamic power spectrum. For example, the contribution of the signal power is reported to be around 30% of dynamic power for a 45 nm high-performance microprocessor synthesized using a structured data paths design style and about 18% of the overall power spectrum [1].

The signals are complex structures in nanometer technologies that span over many metal layers. The power of a route segment depends on its width, metal layer, and spacing relative to its adjacent parallel-running routes. These factors determine the area, fringe, and coupling capacitances which impact power. Furthermore, in MSV designs, the power of a routed net depends on its corresponding supply voltage. For example, a route will have lower power if all its terminal cells have (the same) lower supply voltage. If a net connects a driver cell of lower voltage to a sink cell of higher voltage, its route includes a level converter (LC) and is decomposed into two segments of low and high supply voltages, corresponding to before and after the LC.

Figure 1 shows an example to motivate for power-aware global routing. Three nets are given in this example with a corresponding power supply (V_L low voltage or V_H high voltage). An activity factor is also given for each net. A net with higher power supply and activity factor consumes more power. Figure 1(a) shows that a shortest-length routing results in overflow in routing resources. The congested area is shown in the figure. Traditional GR is based on minimization overflow with minimal increase in wirelength. It is shown in this example in Figure 1(b), in which net n_2 is now 2 units longer; however, the congested area is eliminated. However, net n_2 has the highest power consumption (due to higher values of supply and activity factor). Making n_2 longer further increases its power consumption. Therefore, in power-aware GR which is shown in Figure 1(c), net n_2 has the shortest length, however, net n_1 instead is detoured to eliminate



(a) Shortest-length GR results in overflow (OF)

(b) Wirelength-based GR trades off increase in wirelength (WL) with decrease in OF

(c) Power-aware GR makes the high power consuming net shorter with minimal increase in WL and decrease in OF

Figure 1: Motivation for power-aware GR.

congestion. The wirelength of power-aware GR is higher than wirelength-based GR, but it has less signal power.

In this work, we propose a global routing (GR) method that optimizes the signal power in MSV designs. Figure 2 shows a generic design flow for a MSV-based GR. After placement and voltage assignment, the location and supply voltage of each cell are known. The supply voltage is determined either through voltage island generation [2, 3] or through a row-based assignment in a standard cell methodology. Furthermore, LCs are added to any net that connects a driver cell to a set of sink cells of higher supply voltage. Next, GR is applied to minimize the overall wirelength (WL), where the LCs are also included as terminals of a net.

For a given WL-optimized GR solution, we propose to further detour the nets in order to optimize the signal power. The signal power can be approximated during GR since at this stage the metal layers of each route segment are known. Furthermore, the spacing of parallel routes can be estimated from the routing congestion. Given a WL-optimized solution, the nets can be rerouted to trade off WL with power. For example, nets from higher metal layers can be routed to the lower ones for less wire widths and area capacitance. Nets can also be rerouted to spread the congestion, thereby increasing their spacing for less coupling capacitance. Activity factor and voltage can be incorporated as a power-weight for each route.

We present a mathematical formulation for MSV-based GR to minimize power and present integer programming-based techniques to solve the formulation. As part of power saving, our methods spread the routing congestion and ensure no additional overflow (of routing resources) and a bounded degradation in WL compared to the initial solution.

To the best of our knowledge, this is the first work of power-driven global routing in MSV designs. Recently, the work [4] discusses power-driven GR; however it does not consider the MSV case. Furthermore, it relies on the availability of *power-efficient candidate routes* for each net but generates such candidate routes quite heuristically. As part of the contributions of this work, we show a formal procedure to generate power-efficient candidate routes from the initial WL-optimized solution while taking into account the overall WL degradation and power saving. Also, recently the work [5] studies the GR problem for MSV domains, but it does not focus on routing for power minimization.

2. New Algorithmic Techniques Used

Power-aware routing can be considered as a new EDA problem. This is because the power of global interconnects (or signals) are starting to show nonnegligible contribution to the overall power spectrum for advanced technology nodes [1]. This issue is further exacerbated for multisupply voltage domains for which the power of a net dramatically changes depending on the voltage domain(s) that it (fully or partially) falls in. This work is the first to propose and formulate the power-aware routing for multi-supply voltage domains.

Furthermore, from an algorithmic perspective, the techniques offered in this work are a combination of integer programming with parallel processing based on problem decomposition. Integer programming allows obtaining a higher-quality solution compared to using heuristics as shown in [6]. However, it is not considered a suitable algorithmic venue for large-sized industry circuits. This work relies on decomposition of the routing problem intro smaller-sized and parallel-processed subproblems in order to make the use of integer programming possible for large-sized circuits.

3. Interconnect Modeling

In this section, we discuss an MSV-based GR model. We assume that the level converters (LCs) are placed for some of nets and the supply voltage of each cell is known.

3.1. Interconnect Modeling in MSV Designs. We are given a grid-graph $G = (\mathscr{V}, \mathscr{E})$ model of the GR problem, where each

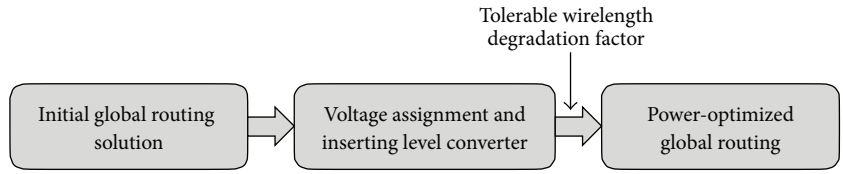


Figure 2: Overview of GR with MSV.

vertex $v \in \mathscr{V}$ corresponds to a global bin containing a number of cells. Each edge $e \in \mathscr{E}$ represents the boundary of two adjacent bins. A capacity r_e is associated with each edge e, reflecting the maximum number of routes that can pass between two adjacent bins. A net $i \in \{1, \ldots, N\}$ is identified by its terminal cells, which are a subset of the vertices $\mathscr{V}$. In MSV-based GR, the terminals of a net may also be the LCs. During GR, a Steiner tree t_i in G is found for each net i to connect its terminals. The length of t_i is taken to be its wirelength (WL).

Figure 3(a) shows an example. The chip is divided into regions. Each region has either a low (V_L) or high (V_H) supply voltage. A routed net is specified in the figure. The net has one driver terminal with V_L voltage and three sink terminals of V_H voltage. The route includes two LCs which are also considered as additional terminals of the net.

For power-driven MSV-based GR, we first decompose a net which contains a LC into a set of subnets. We reroute each subnet as an individual net during power optimization. Consequently, we have $N_d > N$ number of nets after decomposition. For example, in Figure 3(b), the initial route is shown with its LCs. The net is decomposed into three subnets, each of which will be rerouted. The first subnet connects the driver terminal in V_L to the two LCs. The second one connects one LC to one V_H terminal. The third one connects the other LC to the other two V_H terminals.

Figures 3(c)–3(e) illustrate our net decomposition procedure. The decomposition of each net is done using its initial route and the location(s) of its level converter(s), assuming they are determined before this stage. For a net containing level converters, starting from its driver terminal, a subnet corresponding to a low supply voltage is formed that connects the driver terminal to a set of level converters and/or a set of sink terminals of the same supply voltage. Next, one or more subnets are formed that connect the level converters to the sink terminals of the same (and higher) voltage level. The BFS algorithm is utilized to traverse the initial route in our implementation. For example, in Figure 3(d), we start traversing from the source node until reaching the two level converters. All the touched edges form the first subnet n_1 which has a low supply voltage. Next, we continue traversing from each of the level converters individually until reaching all the sink nodes, using which the subnets n_2 and n_3 with high supply voltage are then identified.

Our net decomposition procedure is able to find a minimum number of subnets for each net that contains a level converter such that each subnet has only one corresponding supply voltage. Note that after rerouting the subnets, it is possible that these subnets may pass through the same edge(s) as shown in Figure 3(e). If the subnets which pass through the same edges have the same voltage level, (e.g., the subnets n_2 and n_3 in Figure 3(e)), then we can merge these subnets to release the overutilized routing resources. The above procedure is given for the case when two supply voltages V_L and V_H exist, which is also the case considered in this work. For higher number of voltage domains, the procedure can be extended in a similar way.

3.2. Power Modeling. Each decomposed net $i \in \{1, \ldots, N_d\}$ has a corresponding supply voltage V_i and switching activity α_i. The required interconnect power for a GR solution is estimated as

$$P = f_{clk} \times \left(\sum_{i=1}^{N_d} \alpha_i V_i^2 \left(C_i^{\text{sink}} + C_i^{\text{route}} \right) \right), \tag{1}$$

where f_{clk} is the frequency. As seen in (1), the capacitance of routed net i is the sum of the capacitances of its sink cells (denoted by C_i^{sink}) and of its route (denoted by C_i^{route}). Here C_i^{sink} is a constant that does not depend on the rerouting, so it is excluded from the optimization. Note that the power of the LCs are considered fixed and thus also not considered as part of the interconnect power optimization. The capacitance C_i^{route} for a routed net i is the sum of the capacitances of its unit-length edges that are contained in route t_i (given by notation $e \ni t_i$):

$$C_i^{\text{route}} = \sum_{e \ni t_i} C_e^u. \tag{2}$$

The parameter C_e^u is the capacitance of one routed edge $e \in \mathscr{E}$. This capacitance is a function of the metal layer l_e, wire width w_e, and wire spacing s_e of the edge e. Specifically,

$$C_e^u = Ca\left(l_e, w_e\right) + 2Cf\left(l_e, w_e, s_e\right) + 2Cc\left(l_e, w_e, s_e\right), \tag{3}$$

where Ca and Cf are the area and fringe capacitances with respect to substrate, and Cc is the coupling capacitance. As indicated, these capacitances are functions of wire length, width, and spacing and are provided by the technology library through a lookup table.

In this work, we assume that only one (and a different) wire width is associated with each metal layer, so we exclude the parameter w_e, and for each edge $e \in \mathscr{E}$, its metal layer l_e is known. The spacing for edge e is estimated from the edge utilization u_e in a GR solution. Given the utilization u_e and the length of edge e (computed from the chip dimension and the routing grid granularity), the spacing s_e is calculated to allow maximum spacing between its corresponding routes. Figure 4 shows an example for $u_e = 3$. This simple *averaging* strategy may be adjusted if more information is available at

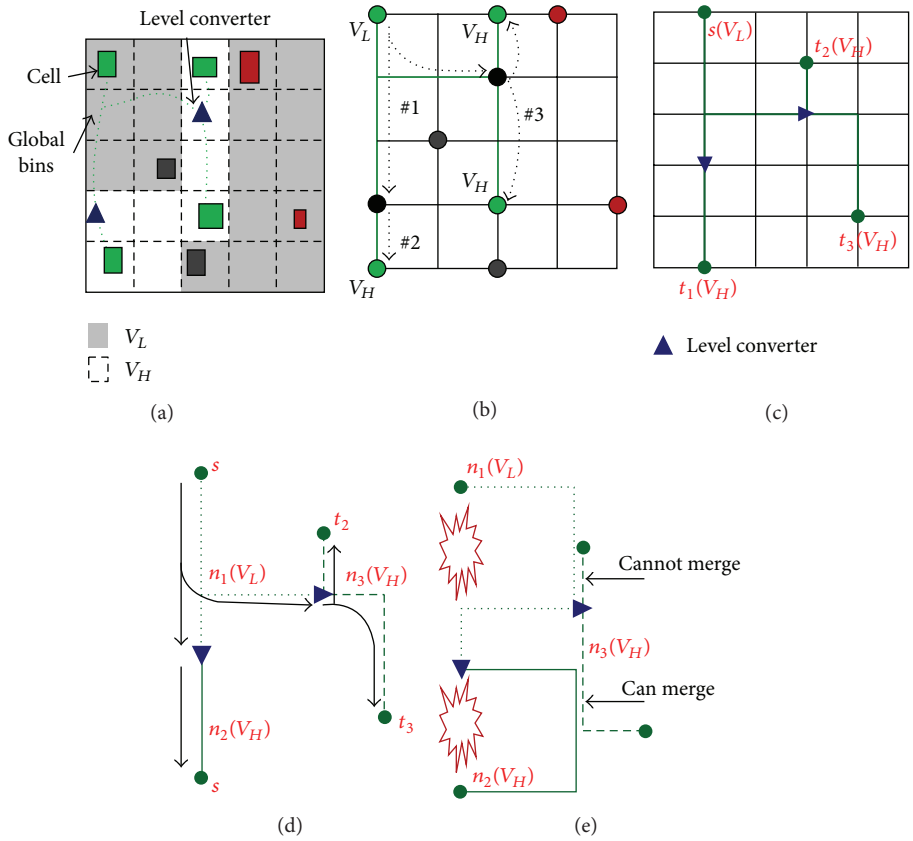


FIGURE 3: Graph modeling and net decomposition in global routing with level converters.

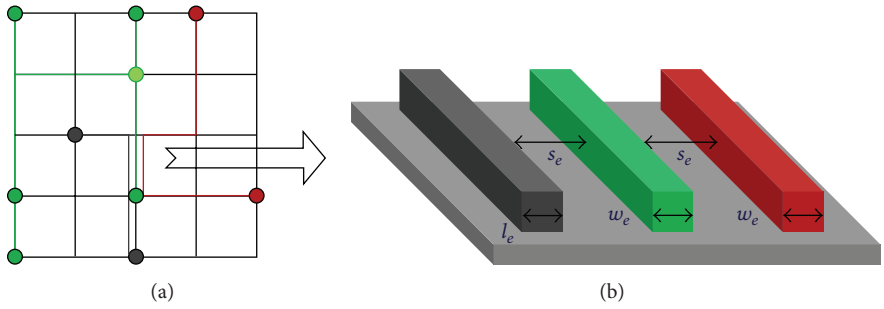


FIGURE 4: Modeling route capacitance on a GR edge.

the GR stage (e.g., the adjustment may be due to the fixed short nets which fall inside a single global routing bin). With this approximation, we can express the capacitance of a unit-length route edge in terms of the edge's metal layer and its utilization. The total capacitance of edge e is given by the product of the per-unit capacitance C_e^u and the utilization u_e: $C_e = C_e^u \times u_e$.

Figure 5(a) shows the curves representing area, fringe, and coupling capacitances for metal layer 1 with respect to edge utilization for a 45 nm library [7], assuming each GR edge is 2μ. The summation of the 3 capacitances (C_e^u) is shown in Figure 5(b).

4. Placement of Level Converters

The LCs may only be placed on the WL-optimized route, initially provided for each net. This ensures that the addition of LCs will not cause extra congestion; it allows connecting each LC to the initial route conveniently just by adding vias

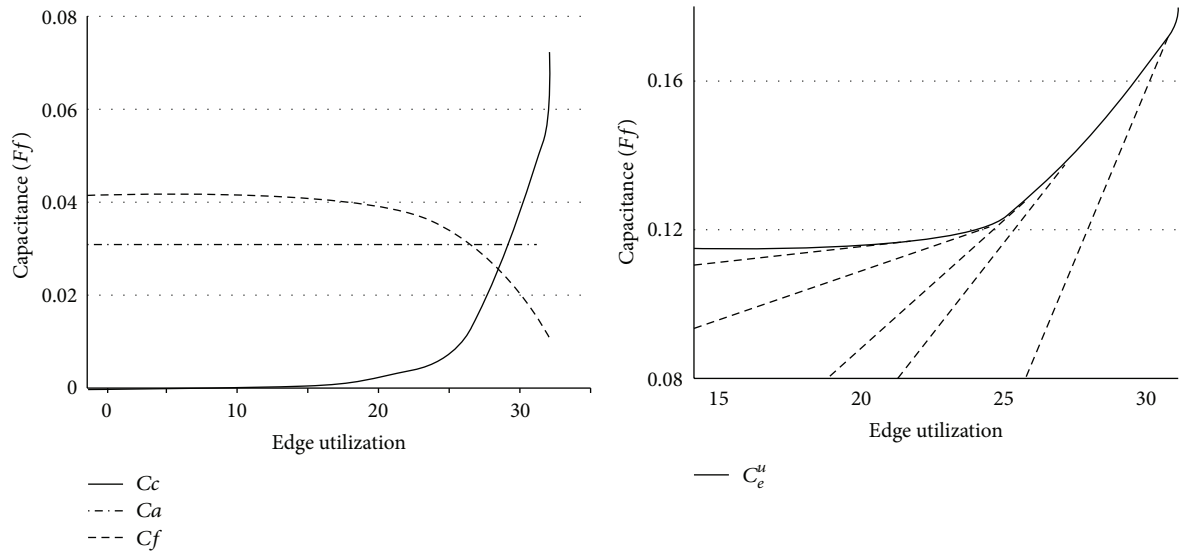


Figure 5: Dependence of three types of capacitance on edge utilization in metal layer 1.

from the LC to the initial route. Randomly placing the LCs may harm the GR congestion and degrade WL or overflow.

We list a set of requirements to identify valid LC insertion cases for a net i with given route t_i. We assume the net has a single source and may have multiple sink terminals.

(1) The location of LC is vertex v in $t_i (v \ni t_i)$.

(2) This vertex v should fall inside a V_H voltage island.

(3) The global bin corresponding to v should have enough space to add the LC. We denote the available space of v by A_v and compute it after placement (see Figure 2).

(4) For k vertices $v_1, \ldots, v_k$ satisfying the above 3 conditions, if all have the same distance to the source terminal (in terms of the number of edges on t_i), we require k LCs be added on these vertices simultaneously.

Figure 6 shows the set of potential LC locations of net i with initial route t_i. The source is the terminal in V_L island. Note that one vertex in t_i cannot be used because it is in the V_L island. We have four cases for valid LC insertion indicated by i_1, i_2, i_3, and i_4. In the latter case, two LCs should be placed on the net after the diverging point on the route to ensure that V_H is delivered to both sink terminals. For a single-source net i, we identify all the cases for valid LC insertion using a breadth first traversal on t_i and denote this set by $\mathscr{L}_i$. In this example $|\mathscr{L}_i| = 4$. For each case $l \in \mathscr{L}_i$, we further compute a corresponding power p_{il} using (1), where the edge utilization required to compute coupling capacitance is obtained from the initial WL-optimized solution. The power includes the interconnect portions on t_i and the LC(s).

To select one LC insertion case for each net, we define binary variable x_{il} to be equal to 1 if and only if case $l \in \mathscr{L}_i$ is selected for net i. The LC placement problem is expressed as the following integer program (IP) which can efficiently be solved using a solver, as we elaborate in our experiments:

$$\min_{x,s} \sum_{i=1}^{N} \sum_{l \in \mathscr{L}_i} p_{il} x_{il} + \sum_{i=1}^{N} M s_i, \qquad \text{(IP-LC)}$$

$$\sum_{l \in \mathscr{L}_i} x_{il} + s_i = 1, \quad \forall i = 1, \ldots, N,$$

$$\sum_{i=1}^{N} \sum_{l \in \mathscr{L}_i} a_{vl} x_{il} \le A_v, \quad \forall v \in V, \qquad (4)$$

$$s_i \ge 0, \quad \forall i = 1, \ldots, N,$$

$$x_{il} = \{0, 1\}, \quad \forall i = 1, \ldots, N, \ \forall l \in \mathscr{L}_i,$$

where the parameter a_{vl} is equal to 1 if, in case l, an LC is placed at vertex v. The first set of constraints ensures at most one LC insertion case is selected for each net. The slack variable s_i will be positive if there is no available space for placing LCs for net i and is heavily penalized by positive M to maximize the number of placed LCs. The second constraints ensure LCs are placed in the free placement space.

In addition, it may not be possible to place LCs on any vertex v on the GR grid because its corresponding global bin is highly congested. We therefore associate for each vertex v a constant parameter A_v, indicating its available placement space. In our experiments, we calculate this available space for each global bin according to the placement density.

With this assumption, after adding an LC, the initial route can connect to the LC by extending through a set of vias at the LC location. Furthermore, for the WL-optimized tree t_i of net i, the potential locations of LCs are only allowed to be those vertices $v \in t_i$ which fall inside the V_H voltage islands as the LC should get connected to V_H voltage.

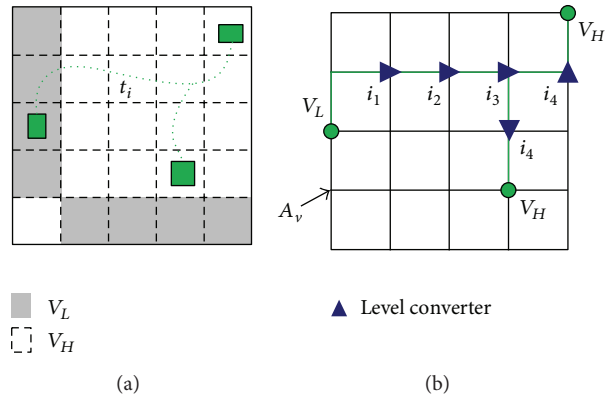


Figure 6: Valid LC locations for one net.

To enumerate all the possible LC insertion cases in a given route, consider a single-source net with WL-optimized route t_i. We enumerate and systematically identify all the cases for *valid* LC insertion according to the distance of vertex v from the source vertex on the route. The distance is measured in terms of the number of edges on the route between v and the source and obtained using breadth first traversal on tree t_i. We count each LC location $v \in t_i$ and in the V_H island as one possibility for LC placement on that route. However, if multiple vertices have the same distance to the source, we consider adding LCs on all such vertices simultaneously and count them as one possibility. For example, in case i_4 in Figure 6 we insert two LCs simultaneously. We then define a set $\mathscr{L}_i$ for each net i, indicating all the possibilities for its valid LC insertion. In the given example $|\mathscr{L}_i| = 4$. Furthermore, we compute the power p_{ij} for LC insertion case j for route i. The power is computed according to (1) where the edge utilizations used to compute coupling capacitance are computed from the provided WL-optimized solution.

5. Power-Driven MSV-Based GR

In this section, we first present a mathematical formulation of power-driven MSV-based GR. We then discuss integer programming-based techniques to obtain high-quality solutions to the formulation.

5.1. Mathematical Formulation. As described in Section 3.2, the per-unit capacitance of an edge $e(C_e^u)$ is a function of its metal layer and the edge utilization. Typically, this function is a convex increasing function, as depicted in Figure 5. We represent the function C_e^u by a set of line segments denoted by Q_e^u. For example, the set Q_e^u is composed of 7 line segments in the library used in this work [7]. Each line segment $q \in Q_e^u$ is of the form $m_q^u + r_q^u u_e$, for a given range of u_e, where m_q^u and r_q^u are derived from the library for that range. For each of the 8 metal layers in our library, the curve C_e^u is represented as 7 piecewise linear segments.

Since the per-unit capacitance is convex, its value may be expressed in our mathematical optimization problem for GR with the following set of linear inequalities:

$$m_q^u u_e + r_q^u \le C_e^u, \quad \forall q \in Q_e^u. \tag{5}$$

For a given edge utilization u_e, the corresponding C_e^u is obtained from the line equation that gives the largest value of $m_q^u u_e + r_q^u$ for $q \in Q_e^u$.

To model GR, we are given a routing grid graph $G = (\mathscr{V}, \mathscr{E})$, a set of decomposed multiterminal nets denoted by N_d, and edge capacities r_e. Let $\mathscr{T}_i$ be a collection of all Steiner trees that can route net i. We later discuss how to approximate $\mathscr{T}_i$ by generating a set of power-efficient candidate trees with consideration of WL degradation. Each tree $t \in \mathscr{T}_i$ is associated with a binary decision variable x_{it} which is equal to 1 if and only if it is selected to route net i. Let the parameter a_{te} be equal to 1 if tree t contains edge e (if $e \ni t$). The GR problem for power minimization is given by

$$\min_{x,s,C^u} \sum_{i=1}^{N_d} \sum_{t \in \mathscr{T}_i} \alpha_i V_i^2 \left(\sum_{e \ni t} C_e^u \right) x_{it} + \sum_{i=1}^{N_d} M s_i, \tag{PGR}$$

$$\sum_{t \in \mathscr{T}_i} x_{it} + s_i = 1, \quad \forall i = 1, \ldots, N_d,$$

$$\sum_{i=1}^{N_d} \sum_{t \in \mathscr{T}_i} a_{te} x_{it} \le r_e, \quad \forall e \in \mathscr{E},$$

$$m_q^u \left(\sum_{i=1}^{N_d} \sum_{t \in \mathscr{T}_i} a_{te} x_{it} \right) + b_q^u \le C_e^u, \quad \forall e \in \mathscr{E},\ \forall q \in Q_e^u, \tag{6}$$

$$\sum_{i=1}^{N_d} \sum_{t \in \mathscr{T}_i} w_{it} x_{it} \le W_0 (1 + \beta),$$

$$s_i \ge 0, \quad \forall i = 1, \ldots, N_d,$$

$$x_{it} = \{0, 1\}, \quad \forall i = 1, \ldots, N_d,\ \forall t \in \mathscr{T}_i.$$

The first term in the expression of the objective function is the interconnect power as explained in Section 3.2. It includes activity α_i and voltage V_i of net i. The capacitance of a route t of net i is obtained by adding the unit edge capacitances C_e^u

for all the edges $e \ni t$. Here, the route $t \in \mathcal{T}_i$ will be selected for net i only if $x_{it} = 1$.

The first set of constraints selects at most one route for each net. The slack variable s_i is equal to 1 if net i cannot be routed, and the variable is penalized in the objective function by a large parameter M to maximize the number of routed nets. The term $\sum_{i=1}^{N_d} \sum_{t \in \mathcal{T}_i} a_{te} x_{it}$ represents the edge utilizations u_e. The second set of constraints ensures that the edge utilizations are within the given edge capacities. The third set of constraints determines the per-unit edge capacitance C_e^u for each edge e from its utilization, using the discussed piecewise linear model. The fourth constraint ensures the new wirelength is within a factor β of the initially-provided wirelength W_0. Here w_{it} denotes the wirelength of route t of net i.

The constraints of formulation (PGR) are all linear. However, the objective expression is nonlinear (due to the multiplication of variables x_{it} and C_e^u). We handle the nonlinearity in a heuristic manner using a two-phase approach. First, we choose a rerouting that attempts to minimize the total capacitance of all edges. Next, per-unit capacitances are estimated (and fixed) based on the solution of the first phase, and a rerouting is sought that minimizes the total estimated power. Each of these two phases becomes integer *linear* programs (IPs) which are discussed in the next sections.

5.2. Phase 1: Minimizing Total Capacitance. Using the piecewise linear approximation for the per-unit capacitance C_e^u given by (5), we may also approximate the total capacitance as

$$C_e = C_e^u \times u_e \geq m_q^u u_e^2 + r_q^u u_e, \quad \forall q \in Q_e^u. \tag{7}$$

This (convex) nonlinear expression may be relinearized, resulting in another piecewise linear expression for the total edge capacitance that may be used in our linear integer program for minimizing the total capacitance:

$$C_e \geq m_q u_e + r_q, \quad \forall q \in Q_e. \tag{8}$$

5.2.1. Formulation. The formulation of phase 1 is given by the following IP:

$$\min_{x,s,C} \sum_{\forall e \in \mathcal{E}} C_e + \sum_{i=1}^{N_d} M s_i, \tag{PGR-P1}$$

$$\sum_{t \in \mathcal{T}_i} x_{it} + s_i = 1, \quad \forall i = 1, \ldots, N_d,$$

$$\sum_{i=1}^{N_d} \sum_{t \in \mathcal{T}_i} a_{te} x_{it} \leq r_e, \quad \forall e \in \mathcal{E},$$

$$m_q \left(\sum_{i=1}^{N_d} \sum_{t \in \mathcal{T}_i} a_{te} x_{it} \right) + b_q \leq C_e, \quad \forall e \in \mathcal{E}, \ \forall q \in Q_e, \tag{9}$$

$$\sum_{i=1}^{N_d} \sum_{t \in \mathcal{T}_i} w_{it} x_{it} \leq W_0 (1 + \beta),$$

$$x_{it} = \{0, 1\}, \quad \forall i = 1, \ldots, N_d, \ \forall t \in \mathcal{T}_i,$$

$$s_i \geq 0, \quad \forall i = 1, \ldots, N_d.$$

The objective expression is similar to formulation (PGR) but the first term is replaced by $\sum_{\forall e \in \mathcal{E}} C_e$ which represents an estimate of the total interconnect capacitance. The third set of constraints is also updated; the variable C_e replaces C_e^u in the previous formulation, and the coefficients in the piecewise linear model are updated to use (8).

5.2.2. A Price-and-Branch Solution Procedure. We approximately solve the (PGR-P1) using the two-step heuristics. First, a *pricing* procedure is used to generate a set of candidate routes for each net that are power-efficient while considering the WL degradation. The pricing step approximates $\mathcal{T}_i$ in the formulation to contain a small set of power-efficient candidate routes, instead of all the potential routes of net i. Second, *branch-and-bound* is applied to solve (PGR-P1), selecting one route for each net from the set of generated candidate routes. The standard branch and bound algorithm can be carried out using a commercial solver. This two-step procedure of generating candidate routes and then running branch and bound is commonly known as price and branch [8, 9]. The price and branch procedure was recently applied to solve the GR problem for WL improvement [6]. We apply the same procedure for power improvement. The major technical difference in our procedure is in the pricing step to find power-efficient candidate routes, which we next discuss in detail.

5.2.3. Overview of Pricing for Route Generation. We solve a linear-programming relaxation of (PGR-P1) by replacing the binary requirements on the variables x_{it} with constraints $0 \leq x_{it} \leq 1$ for all i, for all t. The linear program is solved by an iterative procedure known as column-generation [10]. In column generation, we start by replacing $\mathcal{T}_i$ (set of all possible routes of net i) in formulation (PGR-P1) by subset $\mathcal{S}_i \subset \mathcal{T}_i$, initially containing one candidate route per net. We then gradually expand $\mathcal{S}_i$, adding new routes that may decrease the objective function. Adding the new candidate routes is via a *power-aware pricing condition* for each net.

Before explaining the procedure in more detail, we first give the following notations:

(1) we refer to the LP relaxation of (PGR-P1) in which $\mathcal{T}_i$ is replaced by $\mathcal{S}_i$ and $0 \leq x_{it} \leq 1$ by the "restricted master problem" denoted by (RMLP-P1); the solution of (RMLP-P1) for a given $\mathcal{S}_i$ is denoted by $(\widehat{x}, \widehat{s}, \widehat{C})$;

(2) we refer to the dual of the restricted master problem by (D-RMLP-P1). The solution of (D-RMLP-P1) consists of $(\widehat{\lambda} \leq M, \ \widehat{\pi} \leq 0, \ \widehat{\mu} \geq 0, \ \widehat{\theta} \leq 0)$, corresponding to the dual variables for the first, second, and third set of constraints in the relaxed (PGR-P1), respectively.

The iterative column generation procedure including the pricing condition is enumerated below.

(1) For each net $i = \{1, \ldots, N_d\}$, initialize $\mathcal{S}_i$ with one route. (In this work we start with the solution of [11].)

(2) Solve (RMLP-P1), yielding a primal solution $(\widehat{x}, \widehat{s}, \widehat{C})$ and dual values $(\widehat{\lambda}, \widehat{\pi}, \widehat{\mu}, \widehat{\theta})$ in (D-RMLP-P1).

(3) Generate a new route t^* for net $i = \{1, \ldots, N_d\}$. Using the solution of step 2, evaluate the pricing condition:

if $\widehat{\lambda}_i > \sum_{e \in t^*} \sum_{q \in Q_e} m_q \widehat{\mu}_{eq} - \sum_{e \ni t^*} (\widehat{\pi}_e + \widehat{\theta})$, then $\mathcal{S}_i = \mathcal{S}_i \cup \{t^*\}$.

(4) If an improving route for some net i was found in step 3, return to step 1. Otherwise, stop—*the solution* $(\widehat{x}, \widehat{s}, \widehat{C})$ *is an optimal solution to* (RMLP-P1).

Step 3 gives the pricing condition in terms of the solution of the dual problem (D-RMLP-P1) obtained at the current iteration. This step can determine for a given new route t^* if it should be added to the set $\mathcal{S}_i$ to reduce the objective of (RMLP-P1). However, it does not specify how a new route should be found such that the pricing condition gets satisfied. We discuss a convenient graph-based procedure to generate new route t^* which satisfies the pricing condition.

5.2.4. Route Generation for One Net. To find improving routes for net i, we associate a weight w_e for edge e in the GR grid as

$$w_e = \max_{q \in Q_e} \left(m_q \widehat{\mu}_{eq} \right) - \widehat{\pi}_e - \widehat{\theta}. \tag{10}$$

By the theory of linear programming, for each edge e, at most one dual variable μ_{eq}, $q \in Q_e$ will be positive in an optimal solution to (D-RMLP-P1). Thus, considering route t^*, we can compute the pricing condition as $\widehat{\lambda}_i > \sum_{\forall e \ni t^*} w_e$. We take advantage of this interpretation to identify promising route t^* which satisfies the pricing condition. Given a route $t \in \mathcal{S}_i$ obtained from previous iterations, we obtain t^* by rerouting branches of t with the updated edge weights so that the overall weights of rerouted branches are reduced.

We explain the procedure with the example of Figure 7. Considering two nets a and b, suppose we are initially given the routes t_a and t_b for these two nets. After step 2 at the first iteration of column generation, we obtain edge weights which are given in Figure 7(a). To obtain a new route t_a^* for net a, we reroute different branches of t_a. For each terminal, we identify a branch as the segment connecting it to the first Steiner point on t_a. We then reroute this branch by solving Dijkstra's single-source shortest path algorithm [12] on the weighted graph with the weights of the first iteration, similar to [13, 14]. The route t_a^* is shown in Figure 7(b). After adding t_a^* to $\mathcal{S}_a$, we proceed to the second iteration and obtain new edge weights which are shown in Figure 7(b).

The discussed pricing procedure is similar to [6]. However, it differs in the pricing condition and the way edge weights are set up. For solving (RMLP-P1) and its dual at each iteration, we use the solver CPLEX 12.0. After obtaining the final set $\mathcal{S}_i$, again we use CPLEX 12.0 for the branch and bound step to get the final solution. We further accelerate the process by applying a simple problem decomposition that we will discuss in Section 5.4.

5.3. Phase 2: Considering Activity and Voltage. At phase 2, we approximate the per-unit edge capacitances using the solution from phase 1 and reroute the nets to minimize an approximation of the total power. Since the utilization (and hence capacitance) corresponding to the routing solution of phase 2 may be different from phase 1, we heavily penalize any mismatch in our optimization.

5.3.1. Formulation. We compute the following quantities after phase 1.

(1) We define a new "effective" capacity for each edge e as $\widetilde{r}_e = \sum_{i=1}^{N_d} \sum_{t \in \mathcal{T}_i} a_{te} \widetilde{x}_{it}$, where $\widetilde{x}_{it}$ is the value of the routing solution from phase 1.

(2) We define the new per-unit capacitance as $\widetilde{C}_e^u = \widetilde{C}_e / \widetilde{r}_e$, where $\widetilde{C}_e$ is the value of the edge capacitance from the solution found in phase 1.

With these definitions, the formulation of phase 2 is the following integer linear program:

$$\min_{x,s,\epsilon} \sum_{i=1}^{N_d} \sum_{t \in \mathcal{T}_i} \alpha_i V_i^2 \left(\sum_{e \ni t} \widetilde{C}_e^u \right) x_{it} + \sum_{i=1}^{N_d} M_1 s_i + \sum_{\forall e \in \mathscr{E}} M_2 \epsilon_e, \tag{PGR-P2}$$

$$\begin{aligned}
&\sum_{t \in \mathcal{T}_i} x_{it} + s_i = 1, \quad \forall i = 1, \ldots, N_d, \\
&\sum_{i=1}^{N_d} \sum_{t \in \mathcal{T}_i} a_{te} x_{it} \le \widetilde{r}_e + \epsilon_e, \quad \forall e \in \mathscr{E}, \\
&\sum_{i=1}^{N_d} \sum_{t \in \mathcal{T}_i} w_{it} x_{it} \le W_0 (1 + \beta), \\
&0 \le \epsilon_e \le r_e - \widetilde{r}_e, \quad \forall e \in \mathscr{E}, \\
&x_{it} = \{0, 1\}, \quad \forall i = 1, \ldots, N_d, \ \forall t \in \mathcal{T}_i, \\
&s_i \ge 0, \quad \forall i = 1, \ldots, N_d.
\end{aligned} \tag{11}$$

The first term in the objective expression is summation of an estimate of the power of the nets, where $(\sum_{e \ni t} \widetilde{C}_e^u)$ is the fixed approximate per-unit capacitance of edge e which contains route t and is obtained using the solution of phase 1 as discussed before. The first set of constraints ensures that at most one route is selected per net; otherwise, a heavy penalty of M_1 is associated if $s_i \neq 0$, and this is reflected in the second term of the objective function. The second set of constraints enforces the new utilization of each edge to be $\widetilde{r}_e + \epsilon_e$, where ϵ_e is a new variable which is heavily penalized by a large factor M_2 in the objective function if $\epsilon_e \neq 0$. In other words, we highly penalize if the rerouting of a net causes a larger edge utilization compared to phase 1. This in effect forces the routing process to keep the *mismatch* in the edge utilizations as small as possible which translates in the capacitance (which is function of utilization) to remain close to phase 1. We also enforce $\epsilon_e + \widetilde{r}_e \le r_e$ to ensure that the edge utilization is not beyond its actual capacity r_e in the fourth set of constraints. Finally, the third set of constraints ensures that the increase in wirelength is bounded by factor β.

5.3.2. Solving Using Price and Branch. The solution procedure is quite similar to the one explained in the previous Section 5.2 for phase 1. Here, we just note the differences. We denote the restricted master problem by (RMLP-P2) and its solution by $(\widehat{x}, \widehat{s}, \widehat{\epsilon})$. The dual of the restricted master is denoted by (D-RMLP-P2) and its solution is $(\widehat{\lambda}, \widehat{\pi}, \widehat{\theta})$,

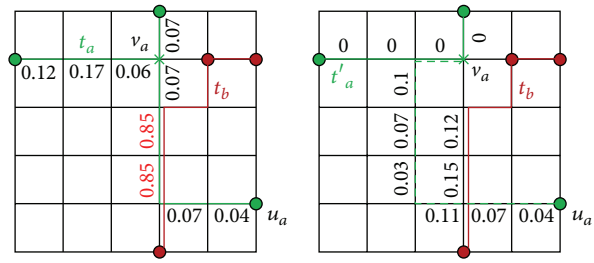


Figure 7: Power-aware route generation.

corresponding to the first, second, and third set of inequalities in relaxed (PGR-P2), respectively.

The initial set $\mathcal{S}_i$ is set to *all the candidate routes generated from phase 1*. This helps to quickly generate a high-quality solution for phase 2. It also ensures that the solution of phase 1 is included as a *feasible* solution in phase 2.

The pricing condition is given by the following inequality $\widehat{\lambda}_i > \alpha_i V_i^2 (\sum_{e \ni t} C_e^u) - \sum_{e \in t} (\widehat{\pi}_e + \widehat{\theta})$ and is used to define the edge weights given by $w_e = \alpha_i V_i C_e^u - \widehat{\pi}_e - \widehat{\theta}$, for all $e \in \mathcal{E}$.

5.4. Decomposition. To accelerate solving the two-phase formulation, we apply a simple problem decomposition. We recursively divide the chip into a set of rectangular subregions while balancing the total number of nets that fall inside each subregion. We use the initial WL-optimized solution of [11] to guide this process. We stop when the number of nets at each subregion is at most 3000, which we empirically determined for our experimented benchmarks from the ISPD2008 suite [15].

Each subproblem is then defined as one rectangular subregion with the set of nets assigned to it. If a net passes from multiple subregions, we force the terminal location on the subregion boundary to be fixed from the initial WL-optimized solution. This allows independent solving of each subproblem without the hassle of later connecting the segments of a route in adjacent subregions. The subproblems are then (one-time) parallel-solved to get the final solution. Figure 8 shows an example.

Even though in our decomposition each subproblem in effect is assigned a low or high voltage level, it is possible that the nets assigned to it have different supply levels. For example, a high voltage net may just pass from a subproblem in a low voltage island, or a net with level converter (which will have portions of high and low voltage levels after net decomposition) may fall in a high voltage island.

Please note, the main difference between our decomposition procedure and [6] is the use of the initial WL-optimized solution to fix the terminal locations on the subregion boundaries and thus avoid later connecting adjacent subproblems.

Overall this decomposition is extended from PGRIP [16], but we make use of our initially provided global routing solution for more effective decomposition to determine the fixed terminal locations on the boundaries for independent and parallel processing of the subproblems.

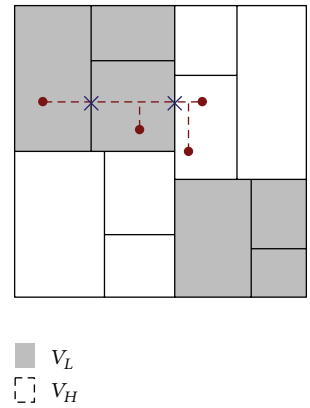


Figure 8: Decomposition into independent subproblems.

6. Simulation Results

6.1. Benchmark Instances. In order to test our solution procedure and determine whether or not significant power savings were possible without increasing wirelength, we modified known benchmarks to include multisupply voltages. Modifying the benchmarks required us to generate timing data and power data and place level converters. We implemented the procedure of [2] to generate voltage islands for two voltage levels of $V_L = 0.9$ V and $V_H = 1.1$ V. The procedure required a sequential netlist with gate-level delay and power models.

Timing Modeling. We assumed the locations of the sequential elements in the ISPD 2008 benchmarks using the following procedure. First, we obtained a directed acyclic graph (DAG) representation of the benchmarks from the variation provided by the ISPD 2006 placement benchmarks [17]. Using the placement benchmarks, we obtained a DAG by starting from the designated primary inputs and traversing in forward direction until reaching the primary outputs. We also assumed the nets with more than 50 terminals to be clock trees to identify sequential elements.

We then assumed that the delay of each cell (or node in the DAG) is proportional to its size (for unit load) where the unit delay was assumed to be of the inverter of the 45 nm library [7] used in this work. We considered loading in our cell delay

modeling to be proportional to the cell size which was also given in the placement benchmarks.

Power Modeling. We randomly and uniformly generated the activity factors of each net to be between 0.1 and 0.9. The 45 nm library used in this work contained information about the total capacitance (area, fringe, and coupling) for each of the 8 metal layers. We used the method described in Section 5 to extract piecewise linear model for C_e and C_e^u for each of the 8 metal layers. For each metal layer, we considered the minimum wire size given in the library. To map edge utilization to spacing, we assumed the length of each edge of the GR grid to be 2μ; for a given utilization we assumed the maximum spacing between the routes mapped to the same GR edge.

Level Converter Placement. After voltage island generation, we needed to decide the locations of the level converters (LCs). (The procedure in [2] did not specify these locations.) For simplicity, we inserted the LCs on the initial WL-optimized solution that was taken from [11]. The LCs were inserted for any net that had a source terminal driving one or more sink terminals. The procedure minimized the number of LCs and placed them as close as possible to the sink terminals, subject to the available whitespace. The whitespace inside each global bin was derived by evaluating (both) the placement and GR variations of the ISPD benchmarks.

6.2. Level Converter Placement. In our first experiment, we report the result from our level converter placement algorithm for the nets that contained a level converter (had a source terminal in V_L island with fanout terminals in V_H islands). We consider the following case in our experiment: we routed all the nets using the initial wirelength-optimized solution of NTHU-Route2.0 [11]. We solve our formulation (IP-LC) to obtain the level converter locations subject to the area density constraints. We consider the obtained results as the base case for power comparison in our second experiment.

Recall the placement of level converters can impact the power of each route by decomposing it into multiple segments where each segment has a high or low supply level. Using (1), we compute the total power of the nets which need level conversion. This includes the power of level converters and the different routes segments of the decomposed nets after inserting the level converters.

Table 1 reports our power comparison results. We report the total number of nets and the number of nets which require level conversion in columns 2 and 3, respectively, for each benchmark. The total number of level converters in our case is given in column 4. The number of level converters is larger than column 3, indicating that for some nets it may be better to add extra LCs but place them closer to the sink terminals to reduce the route portion that is driven by high voltage and save power. In column 5, we report the power of ([11] + LC) for the nets including the ones with level conversion. We use these power numbers as the base case for our next experiment. Finally, the wall clock time of the level converter placement (indicated by WCPU) is given in column 6. As can be seen this step is done very quickly.

6.3. Power-Aware Global Routing. In this experiment, we used the initial WL-optimized solution of [11], and after fixing the locations of LCs, we applied net decomposition (as described in Section 3.1). We then solved two IPs corresponding to the formulations given in phase 1 and phase 2 for each subproblem using CPLEX 12.0 [18]. The number of subproblems is listed in Table 2 column 5 (indicated by SP number) which ranged from 130 to 670 among the benchmarks. These IPs were solved on the computer-aided engineering (CAE) grid at the University of Wisconsin Madison. Each machine had 2 GB of memory. All IPs were submitted to HTCondor [19] which manages the computers in a shared environment. HTCondor then assigned the jobs for parallel processing to the available machines.

Table 2 reports the number of nets, decomposed nets, and LCs in columns 2, 3, 4, and respectively. We then applied our power-driven GR procedure using a wirelength degradation factor of $\beta = 0$, so *no* wirelength degradation was allowed.

We then compared three routing solutions:

(i) the initial WL-optimized solution of [11];

(ii) the solution after applying phase 1, obtained by solving the formulation (PGR-P1);

(iii) the solution by further applying phase 2, obtained by solving (PGR-P1) followed by (PGR-P2).

For each case, we report the wirelength (WL), the total capacitance (C) ($\sum_{i=1}^{N_d} C_i^{\text{route}}$, where C_i^{route} is defined in (2)), given in units fF, and the GR power metric P from (1), excluding the constant portions of the expression.

The results are reported in Table 2 in columns 6 to 14. For the initial solution, we report the wirelength (W_0) of the NTHU-R2.0 routes that have been augmented with the extra via-only segment(s) to connect the LC(s) to the original routes. (As a result, there is slight increase in wirelength compared to the numbers reported in the work [11].) For the solutions of phase 1 and phase 2, we report only the percentage *improvement* in WL, C, and P, all with respect to the initial solution.

As can be seen, applying phase 1 of the power-reduction heuristic results in significant saving of 8.77% in P. Recall, the savings are solely due to capacitance reduction (as can be seen from the higher improvement rate in C compared to P). By further applying phase 2, we see additional improvement in P (on average 16.70%). The improvement in C is slightly larger than phase 1, even though phase 1 solely focuses on optimizing C. This is because we start phase 2 by including all the candidate routes generated from phase 1. Notice that in both phase 1 and phase 2 there is an improvement (reduction) in WL compared to W_0. It is important to note that no extra overflow was introduced in the power-optimized solutions.

In our simulations, we explicitly bounded the runtime for phase 1 and phase 2. The wall clock runtime of all benchmarks for phase 1 and phase 2 was set to 30 min and 40 min,

Table 1: Results of the level converter placement for the ISPD 2008 benchmarks.

Bench	Net number	Net_{LC} number	LC number	Power	WCPU (min)
Adaptec1	177K	9K	20K	432242	5
Adaptec2	208K	8K	17K	336881	7
Adaptec3	368K	17K	43K	1056778	8
Adaptec4	401K	16K	36K	751120	13
Adaptec5	548K	32K	85K	1199591	11
Newblue1	271K	75K	16K	318922	10
Newblue2	374K	22K	47K	453234	17
Newblue4	531K	38K	79K	927712	9
Newblue5	892K	26K	84K	1469859	14
Newblue6	835K	31K	91K	1367000	17
Newblue7	1647K	28K	72K	2201835	21
Bigblue1	197K	9K	26K	619321	6
Bigblue2	429K	15K	43K	560723	13
Bigblue3	666K	23K	60K	814957	12
Bigblue4	1134K	17K	51K	1254323	15

Table 2: Results for ISPD 2008 benchmarks. The WL is scaled to 10^5. Power and cap. are scaled to 10^3.

Bench	Net number	Net_d number	LC number	SP number	Initial solution ([11] + LC)			Phase 1			Phase 1 + phase 2		
					W_0	C	P	−WL (%)	$-C$ (%)	$-P$ (%)	−WL (%)	$-C$ (%)	$-P$ (%)
Adaptec1	177K	197K	20K	130	54.2	953.3	432.2	0.05	11.70	8.57	0.07	15.48	16.17
Adaptec2	208K	224K	17K	195	53.0	750.0	336.9	0.12	10.34	6.93	0.14	14.57	15.13
Adaptec3	368K	411K	43K	359	132.7	2187.0	1056.8	0.01	11.51	8.67	0.34	13.55	13.94
Adaptec4	401K	437K	36K	296	123.0	1613.8	751.1	0.02	12.16	8.46	0.04	16.92	17.20
Adaptec5	548K	632K	85K	454	158.7	2543.0	1199.6	0.38	8.60	6.08	0.43	10.23	10.88
Newblue1	271K	287K	16K	195	47.0	612.2	318.9	0.11	13.39	9.87	0.22	17.45	18.40
Newblue2	374K	421K	47K	312	77.6	894.9	453.2	0.04	14.19	7.87	0.09	19.20	19.34
Newblue4	531K	610K	79K	462	133.7	1955.4	927.7	0.02	13.39	9.61	0.54	17.45	17.61
Newblue5	892K	975K	84K	658	234.7	3405.3	1469.9	0.89	11.55	6.75	0.86	14.00	13.47
Newblue6	835K	926K	91K	532	180.2	2834.9	1367.0	0.62	12.56	9.35	0.57	16.35	17.80
Newblue7	1647K	1719K	72K	670	360.2	5004.4	2201.8	0.01	15.12	11.20	0.17	19.63	20.93
Bigblue1	197K	222K	26K	152	57.0	1110.4	619.3	0.23	10.68	7.20	0.16	12.17	12.56
Bigblue2	429K	472K	43K	275	92.4	1283.8	560.7	0.14	11.64	7.86	0.10	14.76	14.33
Bigblue3	666K	725K	60K	453	133.0	1664.6	815.0	0.91	15.22	10.99	0.93	20.25	20.31
Bigblue4	1134K	1184K	51K	509	233.0	3006.6	1254.3	0.18	16.03	12.12	0.28	22.31	22.46
Avg.								0.25	12.54	8.77	0.34	16.29	16.70

respectively. The number of processors used for parallel processing of the subproblems was upper bounded by the number of subproblems, for example, up to 130 simultaneously processed jobs in benchmark `adaptec1`; the exact number of parallel jobs is not known and depended on the number of free machines in our computational grid (which depended on the number of users of the grid when the simulations ran) as well as HTCondor's internal procedure to schedule jobs to available resources which considers factors such as user priority and past usage history. Furthermore, HTCondor resource management policy ensured that each machine ran at most one job at each time, so the machines were solely dedicated to solving the subproblems when utilized by us.

In an ideal situation (i.e., a grid which can support simultaneous runs of all the subproblems), the wall clock time of our tool will be 30 min and 40 min (for phases 1 and 2, resp.) for a total sum of 70 min for each of the benchmarks. We note, in this work unlike PGRIP [16], our decomposition procedure creates *independent* subproblems so there will not be any communication between the subproblems.

7. Conclusions

We proposed a formulation for minimizing an interconnect power metric for global routing for design with multi-supply

voltage. Power minimization is after an initial wirelength-optimized solution is obtained. We presented a mathematical formulation which considered power saving opportunities by reducing the area, fringe, and congestion-dependent coupling capacitances at each metal layer, while accounting for the activity and supply voltage of each route segment. We showed significant savings in the power metric for global routing without any degradation in wirelength or overflow.

References

[1] R. S. Shelar and M. Patyra, "Impact of local interconnects on timing and power in a high performance processor," in *ACM International Symposium on Physical Design*, pp. 145–152, 2010.

[2] R. L. S. Ching, E. F. Y. Young, K. C. K. Leung, and C. C. N. Chu, "Post-placement voltage island generation," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD '06)*, pp. 641–646, November 2006.

[3] L. Guo, Y. Cai, Q. Zhou, and X. Hong, "Logic and layout aware voltage island generation for low power design," in *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 666–671, January 2007.

[4] H. Shojaei, T.-H. Wu, A. Davoodi, and T. Basten, "A Pareto-algebraic framework for signal power optimization in global routing," in *Proceedings of the 16th ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED '10)*, pp. 407–412, August 2010.

[5] W.-H. Liu, Y.-L. Li, and K.-Y. Chao, "High-quality global routing for multiple dynamic supply voltage designs," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '11)*, pp. 263–269, November 2011.

[6] T.-H. Wu, A. Davoodi, and J. T. Linderoth, "GRIP: scalable 3D global routing using integer programming," in *Proceedings of the 46th ACM/IEEE Design Automation Conference (DAC '09)*, pp. 320–325, July 2009.

[7] Nangate 45 nm open cell library, 2008, http://www.nangate.com/.

[8] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance, "Branch-and-price: column generation for solving huge integer programs," *Operations Research*, vol. 46, no. 3, pp. 316–329, 1998.

[9] D. G. Jørgensen and M. Meyling, "A branch-and-price algorithm for switch-box routing," *Networks*, vol. 40, no. 1, pp. 13–26, 2002.

[10] G. Dantzig and P. Wolfe, "Decomposition principle for linear programs," *Operations Research*, vol. 8, pp. 101–111, 1960.

[11] Y.-J. Chang, Y.-T. Lee, and T.-C. Wang, "NTHU-route 2.0: a fast and stable global router," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD '08)*, pp. 338–343, November 2008.

[12] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[13] M. Pan and C. C. N. Chu, "FastRoute 2.0: a high-quality and efficient global router," in *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 250–255, January 2007.

[14] J. A. Roy and I. L. Markov, "High-performance routing at the nanometer scale," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 6, pp. 1066–1077, 2008.

[15] "ISPD 2008 global routing contest and benchmark suite," http://www.sigda.org/ispd2008/contests/ispd08rc.html.

[16] T.-H. Wu, A. Davoodi, and J. T. Linderoth, "A parallel integer programming approach to global routing," in *Proceedings of the 47th Design Automation Conference (DAC '10)*, pp. 194–199, June 2010.

[17] ISPD 2006 placement contest and benchmark suite.

[18] CPLEX Optimization, *Using the CPLEX Callable Library, Version 9*, Incline Village, Nev, USA, 2005.

[19] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor—a hunter of idle workstations," in *Proceedings of the 8th International Conference on Distributed Computing Systems*, pp. 104–111, 1998.

Hindawi Publishing Corporation
VLSI Design
Volume 2013, Article ID 474601, 15 pages
http://dx.doi.org/10.1155/2013/474601

Research Article

Fast and Near-Optimal Timing-Driven Cell Sizing under Cell Area and Leakage Power Constraints Using a Simplified Discrete Network Flow Algorithm

Huan Ren and Shantanu Dutt

Department of ECE, University of Illinois at Chicago, Chicago, IL 60607, USA

Correspondence should be addressed to Shantanu Dutt; dutt@ece.uic.edu

Received 24 May 2012; Revised 6 November 2012; Accepted 21 November 2012

Academic Editor: Gi-Joon Nam

Copyright © 2013 H. Ren and S. Dutt. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

We propose a timing-driven discrete cell-sizing algorithm that can address total cell size and/or leakage power constraints. We model cell sizing as a "discretized" mincost network flow problem, wherein available sizes of each cell are modeled as nodes. Flow passing through a node indicates the choice of the corresponding cell size, and the total flow cost reflects the timing objective function value corresponding to these choices. Compared to other discrete optimization methods for cell sizing, our method can obtain near-optimal solutions in a time-efficient manner. We tested our algorithm on ISCAS'85 benchmarks, and compared our results to those produced by an optimal dynamic programming- (DP-) based method. The results show that compared to the optimal method, the improvements to an initial sizing solution obtained by our method is only 1% (3%) worse when using a 180 nm (90 nm) library, while being 40–60 times faster. We also obtained results for ISPD'12 cell-sizing benchmarks, under leakage power constraint, and compared them to those of a state-of-the-art approximate DP method (optimal DP runs out of memory for the smallest of these circuits). Our results show that we are only 0.9% worse than the approximate DP method, while being more than twice as fast.

1. Introduction

In order to achieve a balance between design quality and time-to-market, cell library-based design is becoming the dominant design methodology over the custom design method even for high-performance ICs. Usually in a cell library, several different cell implementations are available for the same function with different sizes, intrinsic delays, driving resistances, and input capacitances. Choosing the cell with an appropriate size, that is, *cell sizing*, is a very effective approach to improve timing.

The cell-sizing problem has been studied for a long time. Many methods [1, 2] assume the availability of a continuous range of cell sizes; that is, the size of a cell can take any value in a range. Then, the obtained gate size is rounded to the nearest available size in the library. However, a large number of realistic cell libraries are "sparse," for example, geometrically spaced instead of uniformly spaced [3]. Geometrically spaced gate sizes are desired in order to cover a large size range with a relatively small number of cell instances. Also it has been proved in [4] that, under certain conditions, the set of optimal gate sizes must satisfy the geometric progression. With a sparse library, the simple rounding scheme can introduce huge deterioration from the continuous solution, which often causes the sizing results to fail to meet given timing requirements [3].

On the other hand, few time-efficient methods are known that they can directly handle timing optimization with discrete cell sizes since this problem is NP complete. The technique in [5] uses multidimensional descent optimization that iteratively improves a current solution by changing the size choices of a set of cells that produces the largest improvement. It is not clear how well this method can avoid being trapped in a local optimum. In [3], a new rounding method is developed based on an initial continuous solution. Instead of only rounding to the nearest available size, the method visits cells in topological order in the circuit, and tries several discrete sizes around the continuous solution

for each cell. In order to reduce the search space, after a new cell is visited, it performs a pruning step that discards obviously inferior solutions and a merging step that keeps only several representative solutions within a certain quality region. The run time of this method is significantly larger than the method in [5].

More recently, [6] has proposed a method that uses a combination of continuous and discrete optimization techniques for the power-driven timing-constrained cell-sizing problem. The problem is first simplified to an unconstrained problem using Lagrange relaxation. Then, the resulting unconstrained discrete sizing problem is solved using dynamic programming. Through Lagrange relaxation-based simplification, it is able to handle complex delay constraints of current industry designs. However, as a continuous convex optimization technique, the quality and convergence of the Lagrange relaxation method is not guaranteed when applied to discrete problems. On the other hand, our proposed method directly handles constraint satisfaction simultaneously with objective optimization in a unified and specially designed mincost network flow model.

In this paper, we propose a network-flow-based method for the discrete gate-sizing problem. In our method, the different size options of a cell are modeled by nodes in the network graph. The flow cost represents the change in the timing objective function value when the cell sizes corresponding to the nodes in the graph which have flows through them are chosen. Hence, by solving a mincost flow in the network graph, near-optimal cell sizing can be determined by choosing cell sizes whose corresponding nodes have the mincost flow through them—the near optimality comes from having to constraint the flow to adhere to certain discrete requirements like going through exactly one size-option node per cell.

By modeling the gate-sizing problem as a mincost network flow problem, we can solve it using standard network flow algorithms, which are very time efficient. Also problem constraints, like the maximum allowable total cell area, can be handled efficiently by making the flow amount proportional to the chosen cell area and using an arc with an appropriate capacity to limit the total flow amount.

However, network flow is a continuous optimization method. Thus when applying it to solve the discrete option selection problem, invalid solutions may be produced. For example, the mincost flow may pass through two sizing options for the same cell. We solve such problems by using *min lookahead-cost* or *max flow* selection heuristics. Network flow has been used to solve various EDA problems including placement [7–9] and placement legalization [10]. In the recent technique of [10], only the network graph modeling the legalization problem, and which nodes (representing bins of cells) are overfull (these are then supply nodes) and under full (these are demand nodes) have been determined a priori to solve the mincost flow problem. Costs of arcs, which represent shipment of cells, between adjacent bins are not known in advance since this depends on the exact set of cells being shipped from one bin to an adjacent one. This set of cells and the cost of arcs out of the "current" bin are determined dynamically within Dijstra's shortest path computation that is used iteratively to solve an approximate mincost flow problem. Our problem in using network flow to solve the cell-sizing problem is different: while we know the exact cost and capacities of arcs in the network graph modeling the problem, the issue we need to tackle is that of preventing splitting of the flow among each subset of arcs that represent the selection of the sizes of each cell, so that flow can go through only one of these arcs, thus providing a unique selection of a size for each cell. We solve this problem in an outer loop enveloping the mincost flow computation, as opposed to the technique in [10], which solves it within an inner loop of the mincost flow computation.

This paper is an extended version of the workshop paper [11] that appeared in a internal workshop compendium of papers, which is not considered a published proceedings. Thus it is not strictly necessary to discuss the issue of extensions of that paper. However, there are extensions, which include an updated discussion of previous work, results for benchmark circuits using Synopsys's 90 nm library (in addition to the 180 nm library used in [11]), and an analysis of the differences in the two sets of results for the two libraries.

The rest of the paper is organized as follows. Section 3 provides an overview of our method. A general view of our size selection network graph (SSG) is presented in Section 4. In Sections 5.1–5.4, we discuss various issues of the SSG. In Section 6, we show how to obtain a valid mincost flow in the SSG. Section 7 briefly describes an optimal exhaustive search method to which we compare our network-flow-based technique. Section 8 presents experimental results and we conclude in Section 9.

2. New Algorithmic Approach Used

In this paper we use a simplified version of *discretized network flow* (DNF) that has been recently introduced as a versatile optimization technique for CAD problems over the last four years [7, 9, 12–14]. The DNF technique imposes certain discrete requirements on the flow through the network graph G like a mutual exclusivity constraint on certain arc sets, called *mutually exclusive arc* (MEA) sets, in which at most one arc can have a nonzero flow in each MEA set. In the above cited works, the DNF problems is modeled as a fixed-charge network flow problem [15] in order to solve complex physical synthesis problems using multiple transforms and constraints with significant efficacy. However, it is possible to solve the discrete cell-sizing problem near optimally with a simple version of DNF in which max-flow or min-lookahead-cost (subsequently termed "mincost" for brevity) heuristics for determining which arc in each MEA set should have nonzero flow. We present this simpler version of DNF as the new algorithmic technique in this paper.

3. Overview of Our Method

Our cell-sizing method starts from an initial sizing solution that may be far from the optimal. The objective is to improve the critical path delay of the circuit by resizing cells. We define $\mathcal{P}_\alpha$ to be the set of paths with a delay greater than $1 - \alpha$

fraction ($\alpha < 1$) of the most critical path delay. In order to reduce the complexity of the problem we only consider changing the sizes of cells in $\mathscr{P}_\alpha$. In our experiments, α is set to be 0.1. This simplification does not limit the optimization potential of our method, since our method is incremental in its nature. Thus we can iterate it several times to take more paths into consideration.

We define $\mathrm{CS}(n_j)$ to be the set of critical sinks of a net n_j, which are (1) all sinks in $\mathscr{P}_\alpha$ if the net is in $\mathscr{P}_\alpha$, or (2) the sink with the minimum slack otherwise. Our method tries to improve the critical path delay by minimizing the objective function proposed in [9]. A timing cost $t_c(n_i)$ of a net n_i is defined as [9]:

$$t_c(n_i) = \sum_{u_j \in \mathrm{CS}(n_i)} \frac{D(u_j, n_i)}{S_a^\beta(n_i)}, \tag{1}$$

where u_j is a sink cell of net n_i, $D(u_j, n_i)$ is the net delay of n_i to u_j, and $S_a(n_i)$ is the *allocated slack* of a net in the initial sizing solution; the allocated slack is defined as the path slack divided by the number of nets in the path. β is the exponent of the allocated slack used to adjust the weight difference between costs of nets on critical and noncritical paths. Based on experimental results, $\beta = 1$ works best in this scenario, since only nets in $\mathscr{P}_\alpha$ and those connected to it are considered in the optimization function (see below). Let $\widehat{\mathscr{P}}_\alpha$ be the set of nets that are either in $\mathscr{P}_\alpha$ or connected to nodes in it. The timing objective function F_t is the summation of $t_c(n_i)$ of all nets in $\widehat{\mathscr{P}}_\alpha$ given as

$$F_t = \sum_{n_i \in \widehat{\mathscr{P}}_\alpha} t_c(n_i). \tag{2}$$

Here we only consider nets in $\widehat{\mathscr{P}}_\alpha$, since the delays of only these nets will change with our selection of resizable cells. Note that in this objective function the nets on more critical paths have a higher contribution since the net cost is inversely proportional to the allocated slack and thus will be optimized more—a desirable outcome, especially in a scenario where there is a quota on the resource (e.g., total area) available for optimization.

Usually, the total area is given as a constraint for timing driven cell sizing. Our algorithm can also handle this constrained optimization problem.

4. Overview of the Size Selection Graph (SSG)

We model the timing-minimization cell-size selection problem as a mincost network flow problem. A network flow graph called *the size selection graph* (SSG) is constructed in which the set of sizing options of each cell is modeled as a set of *sizing option nodes*, and each sizing option node corresponds to one sizing option. We use flows in the SSG to model cell-size selection; that is, a size option of a cell is chosen by a flow in the SSG if its corresponding option node in the SSG is on the path of the flow. Hence, each flow through the SSG that passes through one option node in every set of sizing options in the SSG corresponds to a sizing selection for the cells in $\mathscr{P}_\alpha$. Furthermore, if we can set the costs of arcs between option nodes in the SSG in such a way that the total cost incurred by a flow through the SSG is equal to the change in the timing objective function F_t (2) corresponding to the sizing scheme selected by the flow, then the mincost flow in the SSG selects the optimal cell-sizing scheme for F_t. Hence the problem of finding the optimal cell sizing is converted to the problem of finding the mincost flow in a graph for which several efficient algorithms such as the network simplex algorithm and the enhanced scaling algorithm [16] are available. The general structure of our SSG is described below.

4.1. The SSG Structure. Since F_t is the summation of the timing costs t_c of nets, we employ a divide-and-conquer approach in constructing the SSG; that is, first a mininetwork flow graph called a *net structure* is constructed for each net in $\mathscr{P}_\alpha$, and then net structures of connected nets are connected by *net spanning structures* to form the complete SSG as shown in Figure 1. Thus, the SSG has a similar topology as the paths in $\mathscr{P}_\alpha$. Note that the source node S is the only supply node with total flow of $F(S)$ (whose determination is discussed in Section 5.4), and the sink T is the only demand node of flow amount $F(S)$ in the SSG. A net structure N_j is a *child net structure* of N_i if they are connected, and N_j follows N_i in the flow direction in the SSG (i.e., the signal direction in the circuit is from net n_i to net n_j); correspondingly N_i is also a *parent net structure* of N_j. Each net structure contains the sets of option nodes for cells in the corresponding net.

Let us denote the set of sizing options of a cell u as S_u, and the lth sizing option in it as S_u^l. For a net n_i with a driving cell u_d, the net structure is constructed by connecting each sizing option $S_{u_d}^l$ of u_d to all sizing option nodes in S_{u_k} of every sink cell u_k to form a complete bipartite subgraph between S_{u_d} and S_{u_k}. An example is shown in Figure 2. The net has one driving cell u_d and two critical sink cells u_j and u_k; see Figure 2(a). The corresponding net structure is shown in Figure 2(b). Here, we only show two sizing options in each option set. A flow f through the net structure is also shown, which selects size option $S_{u_d}^1$ for u_d, $S_{u_k}^1$ for u_k, and $S_{u_j}^2$ for u_j. With the complete bipartite subgraphs between the sizing option sets of the drive cell and sink cells in a net structure, every possible combination of size choices of cells in the net has a corresponding flow through the net structure and hence is considered in our size selection process.

There are two major issues that need to be tackled in constructing the SSG in order to correctly map the mincost flow in the SSG to an optimal sizing choice. They are as follow.

(1) In each net structure, the cost of each arc needs to be determined so the total cost of a flow through the net structure can accurately capture the change value of the timing cost t_c for the net corresponding to the sizing options chosen by the flow. A detailed description of this issue is given in Section 5.1.

(2) The flow that is determined must be a valid flow, that is, can be converted to a valid sizing option selection. A valid sizing option selection requires the satisfaction of two types of *consistencies*. First, sizing option nodes of a particular cell

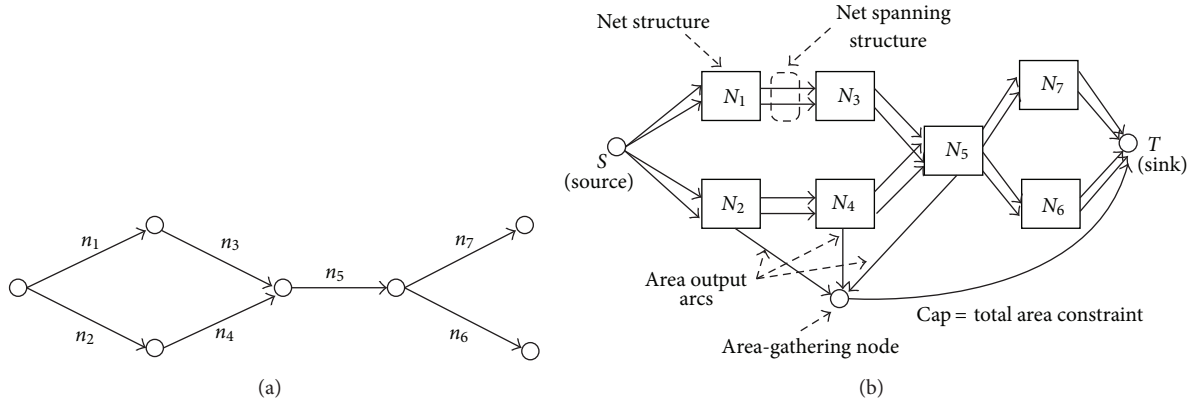


FIGURE 1: (a) Nets in $\mathscr{P}_\alpha$ of a circuit. (b) The corresponding SSG, which includes net structures and net spanning structures. N_i is the net structure corresponding to net n_i. Each net structure will send a flow of amount equal to the total cell size it chooses through the area output arc to the area-gathering node. All these flows are routed to the sink through an arc from the area-gathering node with a capacity equal to the given total cell area constraint.

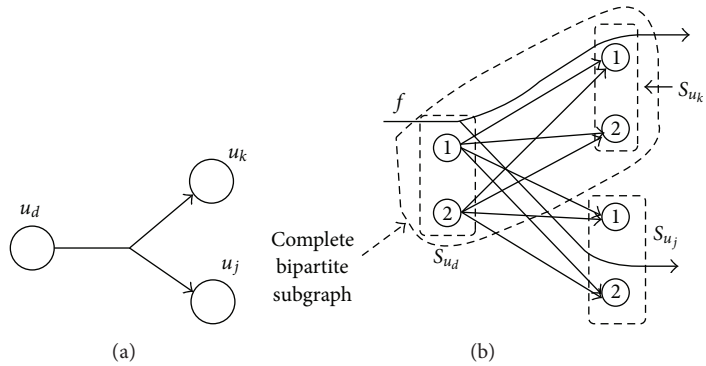


FIGURE 2: (a) A net in critical paths. (b) The net structure corresponding to the net; f is a flow through the net structure.

have to be chosen in a mutually exclusive manner (consistency in a sizing option set). Second, if a cell is connected to multiple nets, its sizing options will also be included in each of the corresponding net structures. In such cases, the selected sizing options for the cell must be consistent across all these net structures (consistency across net structures). As described in Section 5.2, the net spanning structure is designed to guide flows between net structures in the SSG to satisfy the second type of consistency. To guarantee the first type of consistency, two heuristic methods are proposed that prune flows corresponding to invalid option selections when determining the mincost flow; these are discussed in Section 6.

4.2. Handling Cumulative Constraints. We define a *cumulative* metric as one, that is, the sum over all relevant circuit components (cells in our case) of a function $f(s_{i,j})$ of the chosen option (cell size in our case) $s_{i,j}$ of component/cell c_j. A cumulative constraint is then an upper-bound or lower-bound constraint on a cumulative metric. We tackle upper-bound cumulative metrics, specifically, total cell area and total leakage power in this paper.

To handle any given total cell area constraint, each net structure has an outgoing arc called the *area output arc*. A *shunting structure* as explained in Section 5.3 is present in each net structure and connects the area output arc with the option nodes in the net structure. The function of the shunting structure is that when an option node is chosen, the shunting structure diverts a flow of amount that equals the chosen size to the area output arc from the incoming flow through the option node. All these flows are gathered at the *area-gathering node* as shown in Figure 1(b) which is connected to the sink. By setting the capacity of the arc between the area gathering node and the sink to be the given total cell area limit, we make sure the total selected cell area, which is equal to the total incoming flow amount to the area-gathering node, is smaller than or equal to the given limit. The shunting structure is discussed in detail in Section 5.3. Note that, as mentioned before, the sizing option of a cell can be contained in more than one net structure; in this case, the flow amount equal to its selected area will be sent to the area output arc in only one of the net structures that contain the cell.

A leakage power constraint is handled similarly, by having a flow equal to the leakage power of a cell corresponding to its chosen size option go through the shunting structure into a *power-gathering node*, and having an arc from this node to the sink with capacity equal to the leakage power upper-bound constraint.

Algorithm FlowSize
(1) Construct a net structure as depicted in Figure 2 for each net in $\mathscr{P}_\alpha$.
(2) Determine the cost of each arc in the net structures, so that the change in timing cost is accurately incurred by flows through them.
(3) Connect net structures with net spanning structures that maintain consistency of size selection of common cells across multiple net structures; see Section 5.2.
(4) Add the shunting structure and an area output arc (Section 5.3) to each net structure to divert flow of amount equal to the selected size options (nodes) to the area output arc.
(5) Connect area output arcs to the area gathering node that has only one outgoing arc with capacity equal to the area-constraint to limit total selected cell area (= flow amount into the node).
(6) Determine a valid min-cost flow in the resulting SSG by applying the standard min-cost flow algorithm and the min-cost/max-flow heuristics (Section 6) iteratively.
(7) Select the size options chosen by the valid min-cost flow as cell sizes to obtain a near-optimal critical path delay for the circuit.

Algorithm 1: Network flow algorithm for the cell sizing problem for optimizing timing under a given area constraint.

The high-level pseudocode of our cell-sizing method FlowSize is given in Algorithm 1.

5. Further Details of the SSG

In this section, we discuss important details of the SSG, including determining arc costs and capacities, net spanning structures, and further details of the structures for constraint satisfaction.

5.1. Arc Cost Determination. To explain our arc cost formulation that accurately captures the timing cost change, we assume a lumped capacitance and resistance net delay model, which is widely used in cell sizing [2, 4, 17]. For net n_i, the delay $D(u_j, n_i)$ to a sink cell u_j of n_i is

$$D\left(u_j, n_i\right) = R_d\left(c \cdot L_{n_i} + \sum_{u_k \in \mathrm{SC}(n_i)} C_{u_k}\right), \tag{3}$$

where R_d is the driving resistance of n_i, c is the unit WL capacitance, L_{n_i} is the WL of n_i, $\mathrm{SC}(n_i)$ is the set of sink cells of n_i, and C_{u_k} is the input capacitance of cell u_k. In the pre-placement sizing stage, the WL of a net is usually estimated according to the fan-out number of the net. In the postplacement resizing stage, it can be estimated using one of several well-know models, for example, HPBB. With this delay model, for a critical net n_i with m critical sinks, the timing cost $t_c(n_i)$ of n_i is

$$t_c\left(n_i\right) = \frac{m}{S_a^\beta\left(n_i\right)} \cdot R_d\left(c \cdot L_{n_i} + \sum_{u_k \in \mathrm{SC}(n_i)} C_{u_k}\right). \tag{4}$$

In the above expression, let us denote the coefficient $m/S_a^\beta(n_i)$ as α. The parameters affected by cell-sizing options are R_d and C_{u_k}. Hence, if a term in the formula includes R_d, its value is determined by the size of u_d, and if a term includes C_{u_k}, its value is determined by the size of u_k. For example, the term $\alpha R_d \cdot C_{u_k}$ is determined by choices in sizes of both u_d and u_k, and the value change of this term $\Delta\alpha R_d C_{u_k}(S^l_{u_d}, S^m_{u_k})$ when the two sizing options $S^l_{u_d}$ and $S^m_{u_k}$ are chosen can be written as

$$\begin{aligned}&\Delta\alpha R_d C_{u_k}\left(S^l_{u_d}, S^m_{u_k}\right)\\ &\quad = \alpha\left[R_d\left(S^l_{u_d}\right) \cdot C_{u_k}\left(S^m_{u_k}\right) - R_d\left(S'_{u_d}\right) \cdot C_{u_k}\left(S'_{u_k}\right)\right],\end{aligned} \tag{5}$$

where $R_d(S^l_{u_d})$ is the driving resistant corresponding to the driver cell u_d with size option $S^l_{u_d}$, $C_{u_k}(S^m_{u_k})$ is the input capacitance of u_k with size option $S^m_{u_k}$, and S'_{u_d} and S'_{u_k} are the original sizes of u_d and u_k, respectively. The term $\alpha R_d \cdot c \cdot L_{n_i}$ is determined only by the size of u_d, and its value change $\Delta\alpha R_d c L_{n_i}(S^l_{u_d})$ when the option $S^l_{u_d}$ is selected is

$$\Delta\alpha R_d c L_{n_i}\left(S^l_{u_d}\right) = \alpha \cdot c \cdot L_{n_i}\left(R_d\left(S^l_{u_d}\right) - R_d\left(S'_{u_d}\right)\right). \tag{6}$$

We denote the set of arcs between S_{u_d} and S_{u_k} as $E(S_{u_d}, S_{u_k})$, where u_d is the driving cell, and u_k is a sink cell. A valid flow will pass through only one arc in each such arc set, since only one option in S_{u_d} and S_{u_k} can be meaningfully chosen. Hence, in order to make the valid flow cost equal to the change of the timing cost, the cost of an arc in an arc set is set as the sum of the changes of all terms in the timing cost function that is functions of the cell-size options represented by the arc. Furthermore, if a term in t_c is a function of only the size of one cell v, then we arbitrarily choose an arc set $E(S_v, S_w)$ among all those connected to S_v, and the term value change determined by each S^l_v is added to all arcs starting from option node S^l_v in the arc set.

Thus, the value change of term $\alpha R_d C_{u_k}$ is included in the cost of the arc set $E(S_{u_d}, S_{u_k})$. Specifically, the cost of an arc $(S^l_{u_d}, S^m_{u_k})$ in the set includes $\Delta\alpha R_d C_{u_k}(S^l_{u_d}, S^m_{u_k})$. The value change of term $\alpha R_d c \cdot L(n_i)$ is a function only of the driving cell size and thus is included in the cost of arcs in only one arc set $E(S_{u_d}, S_{u_j})$, where u_j is an arbitrary chosen sink cell of n_i. Specifically, the cost of an arc $(S^l_{u_d}, S^m_{u_j})$ in the set includes $\Delta\alpha R_d c L_{n_i}(S^l_{u_d})$.

Finally, the size change of a cell in a critical net also affects the timing cost of noncritical nets that are connected

to the cell. Instead of also constructing net structures for these affected noncritical nets, we use a much simpler method, which is including the timing cost changes of noncritical nets in the net structures of their connected critical nets. Let u be a cell in $\mathcal{P}_\alpha$. We have two cases with respect to u and any noncritical net (a net not in $\mathcal{P}_\alpha$) it may be connected to: (1) if u is the driver of a noncritical net n_j, the timing cost change of n_j is $(R_d(S_u^l) - R_d(S_u'))(c \cdot L_{n_j} + C_{\text{load}}(n_j))/S_a^\beta(n_j)$, where S_u^l is the chosen option of u, S_u' is the initial size of u, and $C_{\text{load}}(n_j)$ is the total load capacitance of n_j. (2) Otherwise, if u is a sink cell of a noncritical net n_j, the timing cost change of n_j is $R_d(C_u(S_u^l) - C_u(S_u'))/S_a^\beta(n_j)$. Let $\Delta t_c^u(S_u^l)$ denote the total timing cost change of all noncritical nets connected to u. If $u = u_d$ is the driving cell of the critical net n_i, $\Delta t_c^u(S_u^l)$ is included in arcs in one arc set $E(S_{u_d}, S_{u_j})$, where, again, u_j is the arbitrarily chosen sink cell of n_i. Otherwise, if $u = u_k$ is a sink cell, $\Delta t_c^u(S_u^l)$ is included in the arc set $E(S_{u_d}, S_{u_k})$. Specifically, the cost of an arc $(S_{u_d}^m, S_{u_k}^l)$ includes $\Delta t_c^u(S_u^l)$.

To sum up, for an arc $(S_{u_d}^l, S_{u_k}^m)$ in a net structure, if $u_k = u_j$ (u_j is the chosen sink in whose arc set $E(S_{u_d}, S_{u_j})$ cost, that is, costs of the arcs in this set, and we include the change in the terms in F_t that are only dependent on the cell size of driver u_d), its cost $(S_{u_d}^l, S_{u_k}^m)$ is

$$\begin{aligned}\text{cost}\left(S_{u_d}^l, S_{u_k}^m\right) = {} & \Delta\alpha R_d C_{u_k}\left(S_{u_d}^l, S_{u_k}^m\right) + \Delta\alpha R_d c L_{n_i}\left(S_{u_d}^l\right) \\ & + \Delta t_c^{u_d}\left(S_{u_d}^l\right) + \Delta t_c^{u_k}\left(S_{u_k}^m\right).\end{aligned} \tag{7}$$

Otherwise if $u_k \neq u_j$, its cost is

$$\text{cost}\left(S_{u_d}^l, S_{u_k}^m\right) = \Delta\alpha R_d C_{u_k}\left(S_{u_d}^l, S_{u_k}^m\right) + \Delta t_c^{u_k}\left(S_{u_k}^m\right). \tag{8}$$

We should note that our cost formulation is accurate under the assumption that the delay is linearly proportional to the load capacitance change. Only under such assumption, for a multiple fanout net, we can sum up the cost for the size change of each fanout to obtain the total delay change of the net. Unfortunately, in modern libraries with small feature sizes, the delay of a cell usually shows a nonlinear relationship with load capacitance. To handle this issue of a nonlinear delay function (as a function of capacitive load) in the ISPD'12 library (where the non-linearity is very pronounced), we determine in each iteration of the mincost flow computation (see Section 6), a min-square error linear approximation around the current design point.

5.2. Maintaining Consistency across Net Spanning Structures. As we mentioned before, the net spanning structure is designed to maintain consistency across net structures. Note that if multiple nets have a common cell, their net structures must be connected in the SSG by net spanning structures.

We first consider the situation in which a cell u is a sink cell of a net n_i, and the driving cell of k nets $n_{j_1}, \ldots, n_{j_k}$ ($k \geq 1$); see Figure 3(a). The size option set S_u is present in all the net structures corresponding to these connected nets. We connect these net structures by adding arcs from each option node in S_u of N_i to the equivalent option node (indicating the same size choice) in the S_u's of $N_{j_1}, \ldots, N_{j_k}$, where N_i is the net structure corresponding to net n_i. The resulting spanning structure is shown in Figure 3(b). With this structure, it is easy to see that if one size option of u is chosen by the flow through N_i, then this flow will also pass through the equivalent option nodes in $N_{j_1}, \ldots, N_{j_k}$. Thus, the required consistency is maintained.

The other situation is that the common cell u is the sink cell of more than one net $n_{i_1}, \ldots, n_{i_l}$ ($l > 1$) and the driver of at least one net n_j, as shown in Figure 3(c). In this situation, we will treat each N_{i_m} ($1 \leq m \leq l$) individually and make the connections between each N_{i_r} and N_j as stated in the first situation. However, in this case the spanning structure cannot guarantee the consistency of the option selected for u in $N_{i_1}, \ldots, N_{i_l}$. We use a *mincost* heuristic to tackle this problem; this is described in Section 6.

The costs of all arcs in the spanning structure are 0.

5.3. Structures for Cumulative Constraint Satisfaction. In this subsection we discuss further details on the structures for satisfying the cell area constraint. As described earlier in Section 4.2, a leakage power constraint is handled by a similar structure.

As we mentioned before, for each net structure, there is an *area output arc*. Once a sizing option node is chosen, a flow with an amount equal to the chosen size needs to be sent to this arc—this flow is then sent to the sink via a common *area-gathering node* and its capacity-constrained arc in order to satisfy the total cell area constraint. The simplest way to achieve this is adding to each option node an arc leaving the net structure to the area output arc with a capacity equal to the option size. Then if enough flow comes into the option node, the desired amount will be sent to the area output arc.

An example is shown in Figure 4. A flow f selects two sizing options $S_{u_d}^l$ and $S_{u_k}^m$ and incurs the corresponding cost of arc $(S_{u_d}^l, S_{u_k}^m)$ in a net structure. $A(S_u^l)$ is the size of an option S_u^l. At each of the two option nodes, a branch of flow diverges a part of f to the area output arc with amount equal to the corresponding option sizes.

However, with this structure, the amount of flow that leaves a net structure is dependent on the option selected and is thus a variable. This is not desirable for determining the capacities of arcs in the spanning structures—the capacities of arcs in a spanning structure from net structure N_i to N_j need to be a constant, that is, independent of the size choices made in N_i by the flow entering N_i. We thus need a structure to diverge a constant amount of the flow that enters N_i; this amount is $\sum_{u \in n_i} A_{\max}(S_u)$, where $A_{\max}(S_u)$ is the maximum size of options in S_u.

Our complete structure for diverting flows to the area output arc is shown in Figure 5. In the structure, the capacity of the arc leaving towards the sink (called the *leaving arc*) from each option node in an option set S_u is set to be $A_{\max}(S_u)$. Therefore, the amount of flow leaving the net structure from any option node $\in S_u$ is always $A_{\max}(S_u)$. However, not all this amount is sent to the area output arc. A *shunting node* is connected to each of these leaving arcs

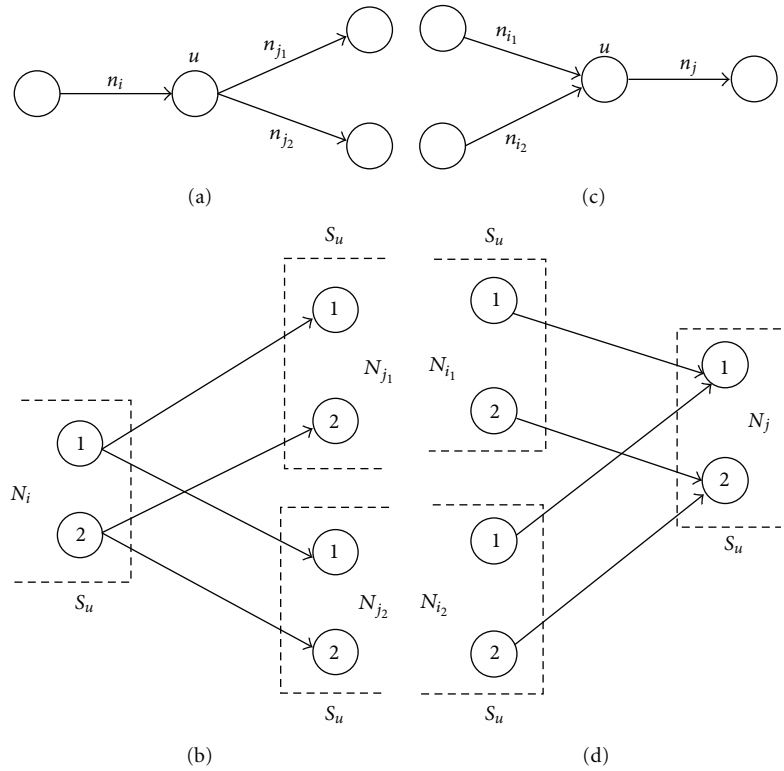


FIGURE 3: The net spanning structure. (a) The common cell u is a sink cell of one net n_i, and the driver of multiple nets n_{j_1} and n_{j_2}. (b) The corresponding spanning structure of (a), where N_k is the net structure corresponding to net n_k. (c) u is a sink cell of multiple nets n_{i_1} and n_{i_2} and the driver of n_j. (d) The corresponding spanning structure of (c).

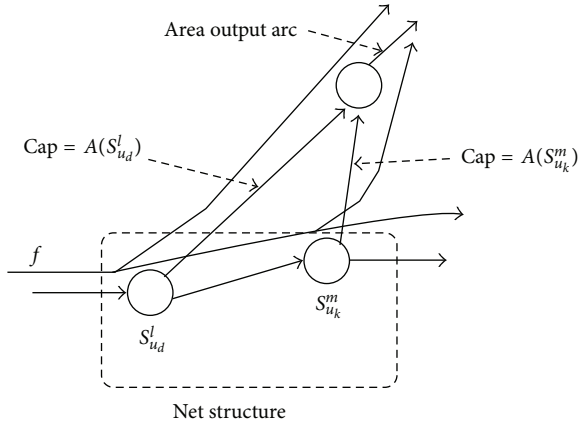


FIGURE 4: Flows to the area output arc of a net structure.

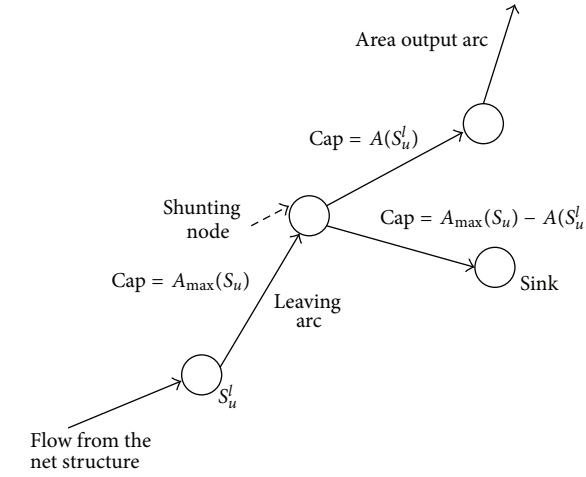


FIGURE 5: The shunting structure for outgoing flows from a net structure.

and divides the flow into two parts, one to the area output arc and the other one *shunted* (i.e., sent directly) to the sink. If the leaving arc is from an option node S^l_u, the capacity of the arc between the shunting node and the area output arc is then $A(S^l_u)$, and the capacity of the arc to the sink is $A_{\max}(S_u) - A(S^l_u)$. Therefore, if S^l_u is chosen, the amount of flow sent to the area output arc is exactly $A(S^l_u)$, and the rest of the amount $A_{\max}(S_u) - A(S^l_u)$ is shunted to the sink. In this way, we send the correct amount of flow into the area output arc of each net structure N_i and also make the total amount of flow leaving N_i to the sink be $\sum_{u \in n_i} A_{\max}(S_u)$. The costs of all arcs in this structure between an option node and the area output arc are 0.

5.4. Arc Capacity Determination. Setting proper capacities for arcs is very important for the correct functioning of the SSG. The capacities should be set such that (1) sufficient flow can be sent to each net structure in the SSG to meet

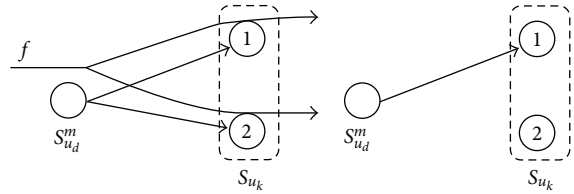


FIGURE 6: (a) Invalid flow f through two sizing options of the same cell u_k. (b) Selecting only one option of S_{u_k} for the next iteration based on some criteria of the previous "split flow."

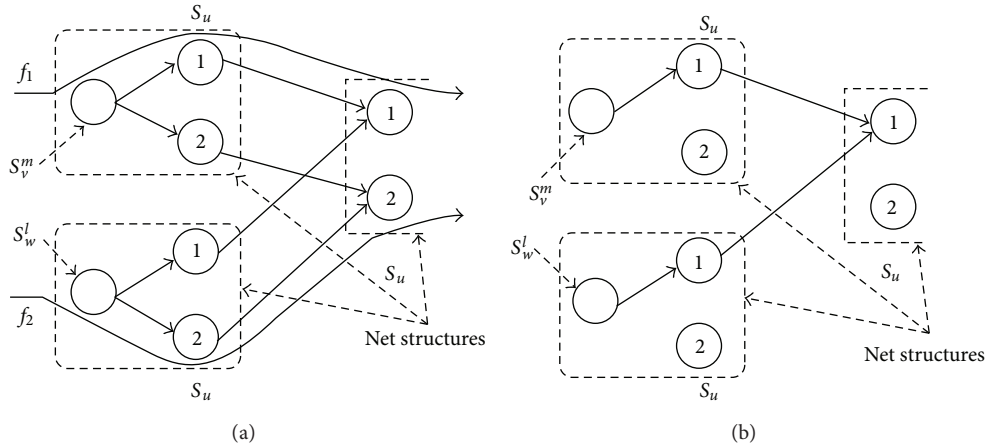


FIGURE 7: (a) Inconsistent option selections of the same cell u in different net structures. f_1 chooses option S^1_u; f_2 chooses option S^2_u. (b) The same option S^1_u of u is selected in the next iteration for all net structures contains u as a sink cell based on some criteria of the previous flow of (a).

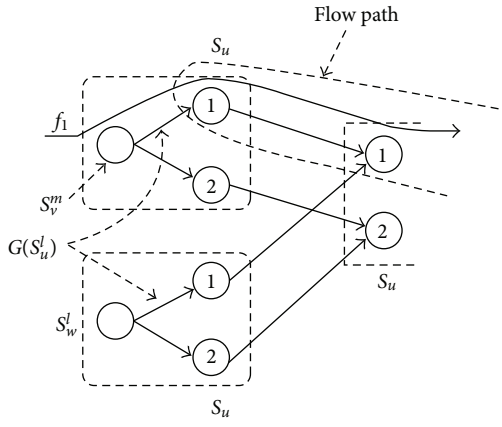


FIGURE 8: The flow path for determining the path cost of S^1_u, and the arcs for determining the arc cost of S^1_u.

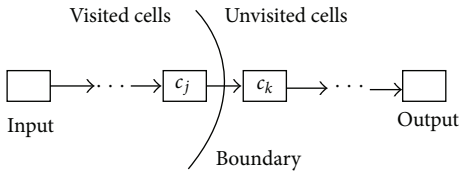


FIGURE 9: Visited cells, unvisited cells, and the boundary between them.

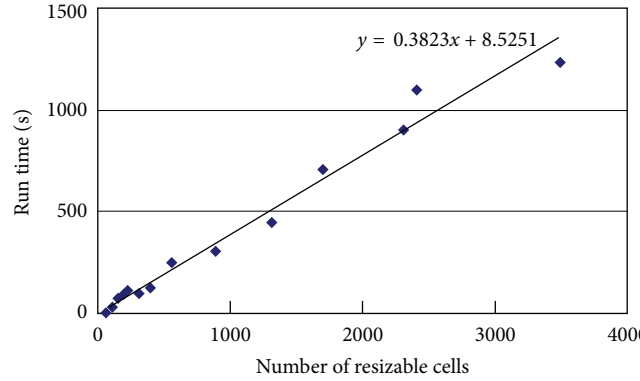


FIGURE 10: Run time versus number of resizable cells.

the flow demand on its area/power output arc; (2) within a net structure, the total incoming flow can be distributed to all sink and the driver cell option sets.

In order to determine the arc capacity, we first determine how much incoming flow is needed for each net structure. A net structure has two types of outgoing flows: (1) into the area output arc for area constraint satisfaction and (2) supply flow to its child net structures. The first type of outgoing flow has a fixed amount $\sum_{u \in n_i} A_{\max}(S_u)$ for a net structure N_i, irrespective of the chosen sizing options, as discussed in Section 5.3. The incoming flow amount must be sufficient to

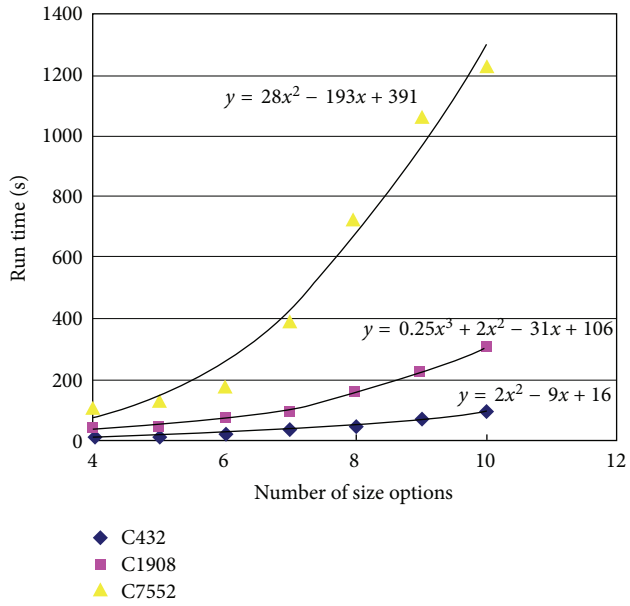


Figure 11: Run time versus number of available size options for each cell for circuits C432, C1908, and C7552. The approximation functions for the empirical time complexity of our method on these circuits are shown.

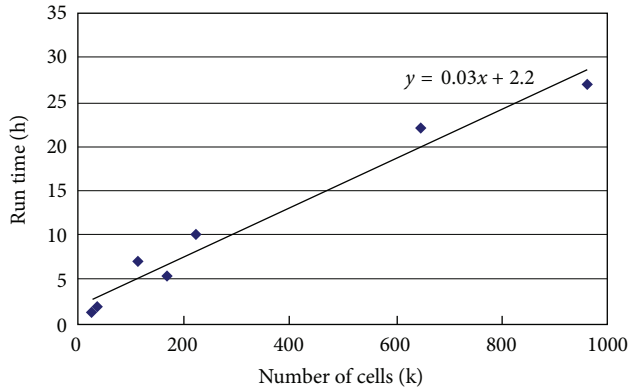


Figure 12: Run time versus number of cells for ISPD 2012 benchmark circuits.

cover the two outgoing flows, and thus the required incoming amount $f_{\text{in}}(N_i)$ of a net structure N_i is recursively given as

$$f_{\text{in}}(N_i) = \sum_{u \in n_i} A_{\max}(S_u) + \sum_{N_j \in \text{child}(N_i)} \frac{f_{\text{in}}(N_j)}{d_{\text{in}}(N_j)}$$

if N_i has any child net structure

$$f_{\text{in}}(N_i) = \sum_{u \in n_i} A_{\max}(S_u),$$

otherwise (N_i) is a "leaf" net structure, (9)

where child (N_i) is the set of child net structures of N_i, and $d_{\text{in}}(N_j)$ is the incoming degree of N_j, that is, the number of parent net structures of N_j. In (9), we assume that the required flow amount for a net structure is sent uniformly from all its parent net structures. The determination of the flow needed in each structure starts from the boundary condition of "leaf" net structures given in (9) that are directly connected to the sink. The incoming flow amount needed for these net structures is the total of their first type of outgoing flows. Starting from the leaf net structures, we visit other net structures in a reverse topological order and determine their required incoming flow amount according to the formulation in (9). The total flow $F(S)$ to be supplied from the source node S is then

$$F(S) = \sum_{N_i \text{ is a "root" net structure}} f_{\text{in}}(N_i), \quad (10)$$

where a root net structure is a net structure of a net that is driven by an I/O cell in the circuit and thus has no "parent" net in the circuit; in the SSG, the parent of all root net structures is S.

After obtaining f_{in} for each net structure, we can determine its arc capacities as follows.

(i) For an arc in the net spanning structure from N_i to N_j, its capacity is $f_{\text{in}}(N_j)/d_{\text{in}}(N_j)$, which equals the flow amount sent from N_i to N_j according to (9). This makes the total incoming flow amount to a net structure N_i exactly $f_{\text{in}}(N_i)$. As mentioned before, the cost of this arc is 0.

(ii) Within a net structure, the capacities of arcs in each arc set $E(S_{u_d}, S_{u_k})$ are the same as is derived below; u_d is the driving cell and u_k is any sink cell of the corresponding net.

For an arc set $E(S_{u_d}, S_{u_k})$, if S_{u_k} is not connected to any arc in the outgoing spanning structure of N_i, the capacity of each arc in it is set as $A_{\max}(S_{u_k})$, so that sufficient flow can be sent to the leaving arc for constraint satisfaction. Otherwise, let $N_{j_1}, \ldots, N_{j_t}$ be the child net structures of N_i that S_{u_k} is connected to (via spanning structures). Note that this means that u_k is a common cell in nets n_i and $n_{j_1}, \ldots, n_{j_t}$. Then, the capacity of each arc in the set is set to be $A_{\max}(S_{u_k}) + \sum_{r=1}^{t} f_{\text{in}}(N_{j_r})/d_{\text{in}}(N_{j_r})$.

(a) *Unit Flow Arc Cost.* When we gave the costs of arcs in a net structure in Section 5.1, we assumed that any flow on the arc will incur the cost. However, in a standard network flow graph, the cost of a flow on an arc is determined as the flow amount multiplied by the unit flow cost of the arc. With the above capacity assignment, a valid flow will always be a full flow on each arc it passes through in a net structure, since the summation of the arc capacity of a single arc in each arc set is equal to the incoming flow amount, and a valid flow uses exactly one arc in each arc set. In order to incur the same cost for a valid flow as determined in Section 5.1, the unit flow cost $C(S^l_{u_d}, S^m_{u_k})$ of an arc $(S^l_{u_d}, S^m_{u_k})$ is set to be the corresponding arc cost given in Section 5.1 divided by its capacity as determined above, that is,

$$C\left(S^l_{u_d}, S^m_{u_k}\right) = \frac{\text{cost}\left(S^l_{u_d}, S^m_{u_k}\right)}{\text{cap}\left(S^l_{u_d}, S^m_{u_k}\right)}. \quad (11)$$

6. Finding a Valid (Discretized) Mincost Flow

The standard mincost network flow solves a linear programming problem (a continuous optimization method). Hence, it cannot automatically handle the consistency (mutual exclusiveness) requirement in a size option set for a particular cell, which may result in an invalid mincost flow for size selection. Therefore, after one iteration of a mincost network flow process, in a net structure, the obtained mincost flow may pass through an option node $S^m_{u_d}$ for the driving cell u_d and then to two or more sizing options in S_{u_k} for a sink cell u_k as shown in Figure 6(a). Furthermore, as explained in Section 5.2, the resulting flow may also violate the consistency requirement for the size option selection across net structures that have a common sink cell.

In the above two cases, we will start a new iteration of the network flow process by pruning out some options that lead to an invalid flow based on certain criteria of the flow so that a near-optimal size selection is obtained. In the new iteration, for the first case, $S^m_{u_d}$ will only connect to one of the option nodes in S_{u_k} that had flow through them in the first iteration as shown in Figure 6(b); the selection criterion is discussed shortly. Similarly for the second case, in all net structures whose corresponding nets have u as a sink cell, the chosen driving cell options in the first iteration, for example, S^m_v and S^l_w as shown in Figure 7(b), will only connect to the same sizing option of u selected from those that had flow through them in the first iteration. In this way, the same invalid flow will not occur in the second iteration.

We have used two alternative selection heuristics to choose a good option node from an illegal selection in each option set S_u to be part of the new iteration.

(i) Max-flow heuristic: always choose the option node with the largest flow amount through it.

(ii) Mincost heuristic: for the first situation, starting from $S^m_{u_d}$, follow the path of each branch of the mincost flow up to a length of l, and choose the option node that is on the branch that has the mincost path.

For the second situation, for each currently selected option of u, the cost that we use to determine whether it is a good option consists of two parts, output path cost and incoming arc cost. As shown in Figure 8, similar to the first situation, the outgoing path cost of an option S^l_u of u is the cost of the flow path starting from the option node. The incoming arc cost of an option S^l_u of u is the total cost of the set of incoming arcs $G(S^l_u)$ to all S^l_u's across all net structures that contain the option set S_u; see Figure 8. The summation of these two costs is a good estimation of the cost of a valid flow that chooses only option S^l_u for cell u. Hence, we choose the option with smallest summation of the path cost and the arc cost. Note that due to run time consideration we cannot always follow each branch flow to the sink. We thus set a limit l (in number of net structures) on the length of paths we follow.

The percentage timing improvement of four representative circuits in Table 1 for the ISCAS85 benchmarks reveals that the mincost heuristics with path length limits of 2, 3, 4, and 5 perform consistently better than the max-flow heuristic and have a relatively better performance in the range of 24–39%. The mincost heuristic is thus implemented in our algorithms, and we set $l = 3$ for a balance between computational complexity and accuracy.

Table 1: Percentage timing improvements of max-flow heuristic and min-cost heuristic with different path length limits.

Ckt	Max flow	Min cost of length			
		2	3	4	5
	%ΔT	%ΔT	%ΔT	%ΔT	%ΔT
C432	8.4	10.0	11.9	12.2	12.2
C1355	4.2	6.0	6.8	6.9	7.2
C3540	12.9	16.1	16.8	17.0	17.1
C7552	8.3	9.3	9.9	10.3	10.5
Average	8.4	10.4	11.4	11.6	11.7

6.1. Time Complexity of FlowSize. It is easy to see that, with the two pruning heuristics, if the option selection for a cell is inconsistent according to the mincost flow obtained in one iteration, in the following iterations the mincost flow will select a valid size option for the cell. Thus, the number of iterations required to reach a valid mincost flow is no more than the total number of cells in $\mathscr{P}_\alpha$.

In each iteration, we use the network simplex algorithm to solve the mincost flow. Given a graph with m arcs, if the capacities and costs of arcs are all integers, with U being the maximum arc capacity and C being the maximum arc cost, then the time complexity of the network simplex method is $O(Um^2 \log C)$ [18]; however, it is well known that the average-case run time of the simplex method is much lower than this worst-case complexity [19]. As described in Section 5.4, the capacities of arcs in our SSG, being the summation of cell sizes, are integers (note that cell sizes in a standard cell design are integer in the unit of the technique feature size of the library). On the other hand, while our cost is not integer, it can be converted to integer values by proper scaling. Hence though we do not actually do the scaling, we use this assumption here in order to derive an upper bound on the time complexity of our algorithm.

Let us first consider the total number of arcs in our SSG. It is dependent on the number of cells N, the number of nets M in the circuit, and the number of available sizes for each cell S. There are three types of arcs in our SSG: arcs between size option sets, arcs in net spanning structures, and arcs in shunting structures. The number of arcs between two size option sets is S^2, and in each net structure there are $d - 1$ sets of such arcs, where d is the degree of the corresponding net. Thus, the total number of these arcs in the SSG is $S^2(d_{\text{avg}} - 1)M$, where d_{avg} is the average degree of nets in a circuit. Since d_{avg} is usually no more than 4, the total number of arcs between size option sets is $O(S^2M)$. The number of arcs in the shunting structure for each option node is three, and hence the total number is $3NS$. The net spanning structure only connects size option sets for the same cell in multiple net structures, and the number of arcs between two of such size

option sets is S. Since the total number of size option sets is $O(N)$, the total number of arcs in the net spanning structure is thus $O(NS)$. To sum up, the total number of arcs in the SSG is $O(S^2M + NS)$.

The maximum arc capacity is equal to the total cell size when all cells are chosen to be at their maximum width and thus is $O(N)$. The maximum arc cost is less than or equal to the maximum net delay. The delay of a net is dependent on the driving resistance of the driving cell, input capacitances of the sink cells, and the net fanout. Since the driving resistances and input capacitances of cells are constants specified by the library and the average fanout is usually a small constant in a VLSI circuit, C can be viewed as a constant.

Typically in a real circuit, N and M are about the same. Hence, the number of arcs in our SSG can be rewritten as $O(S^2N)$. Therefore, the time complexity of each iteration is $O(S^4N^3)$, and the total time complexity is then $O(S^4N^4)$. The polynomially bounded time complexity is a highlight of our algorithm FlowSize, since other discrete cell sizing methods such as [3, 5] are not polynomially bounded in run time. Also, as we show in Section 8 (Figure 10), its actual run time reveals a much smaller complexity, that is, in keeping with the much smaller average-case run time of the network simplex algorithm compared to its worst-case complexity. Furthermore, our experiments show that the actual number of iterations needed in FlowSize is much less than the number of cells in $\mathcal{P}_\alpha$, for example, eight iterations for a circuit with 300 cells in $\mathcal{P}_\alpha$.

7. An Optimal Dynamic Programming Algorithm

Since we do not have the library or library parameters used in [3], we cannot directly compare our results for the benchmarks they use (ISCAS'85) to their published results for those benchmarks. Thus, we implemented an optimal dynamic programming (DP) method that can produce optimal solutions for the sizing problem of cells in $\mathcal{P}_\alpha$ and compared our solution quality to the optimal one.

In the DP algorithm we propose three pruning methods to reduce the number of partial solutions generated in the DP process that can maintain the optimality of the solution but greatly reduce the run time. We process cells in circuit topological order (from driver cells to sink cells). Each time we process a cell, a new set of partial solutions that involve the cell is generated by combining all partial solutions we have for previously processed cells with possible size choices of the cell. The pruning happens after new partial solutions are generated.

We propose three pruning conditions. A partial solution is pruned when (1) it fails to meet the area constraint; (2) it gives longer delay than the critical path delay produced by our method; (3) there is another better partial solution (generated in the search process or extracted from the complete solution of our method) that gives smaller total area, and better arrival time at the outputs of cells on the boundary of the visited region (connected to unvisited cells as shown in Figure 9) in both of the following cases: (a) the unvisited cells are all at their maximum sizes or (b) the unvisited cells are all at their minimum sizes.

The first two pruning methods obviously do not change the optimality of the exhaustive search method. For the third condition, let us first denote the arrival time at the output of a cell c_i as $A_o(c_i)$. Figure 9 shows a single path situation. c_j is the visited cell at the boundary of the visited region, and c_k is the unvisited cell connected to it. We have two partial solutions τ and τ', and τ' is a better solution according to our third pruning condition. Then for any complete solution τ_{comp} expanded from τ, we can also expand τ' to τ'_{comp} by choosing exactly the same sizes for unvisited cells in τ' as in τ_{comp}. Since τ' has smaller area than τ, if τ_{comp} meets the constraints, so does τ'_{comp}. Then the total delay at the output of c_j will be the same for both complete solutions. Since τ' is better than τ, we have $A_o^{\tau}(c_j) > A_o^{\tau'}(c_j)$, where $A_o^{\tau}(c_j)$ is the $A_o(c_j)$ value according to partial solution τ, and $A_o^{\tau'}(c_j)$ is the $A_o(c_j)$ value according to partial solution τ'. Hence, the total delay of the path for τ_{comp} > the total delay for τ'_{comp}. Therefore, τ cannot be expanded to an optimal solution. Thus, our third pruning method also does not negatively affect optimality of the method.

8. Experimental Results

We tested our algorithm on two sets of benchmarks, the ISCAS'85 suite, and the ISPD 2012 suite [21]. For the ISCAS'85 benchmark suite, we used two different libraries, a 0.18 μm (180 nm) library and Synopsys's 90 nm library. We use the same industrial 0.18 μm standard cell library as in [22], which provides four cell implementations for each function with different areas, driving resistances, input capacities, and intrinsic delays. The interval between the four available sizes $w_1 < w_2 < w_3 < w_4$ for each cell is increased about exponentially, that is, $(w_4 - w_3) \approx 2(w_3 - w - 2) \approx 4(w_2 - w_1)$. Other electrical parameters we use are unit length interconnect resistance $r = 7.6 \times 10^{-2}$ ohm/μm and unit length interconnect capacitance $c = 118 \times 10^{-18}$ f/μm. For ISPD 2012 benchmark, it has its own artificial library, which has high nonlinear dependency between delay and load capacitance. Results were obtained on Pentium IV machines with 1 GB of main memory for ISCAS'85 benchmark and Xeon machine with 72 G memory for ISPD 2012 benchmark. Competing methods (the optimal DP method of Section 7 for the ISCAS'85 benchmarks and an approximate DP method [3] for the ISPD'12 benchmarks) were also run on the respective machines.

The ISCAS'85 benchmarks come with initial sizing solutions. We ran our algorithm with a 10% total cell area increase constraint, which means that the total cell area after cell sizing cannot increase more than 10% from the initial solution. The improvements compared to the initial solution obtained by our net work flow method and the optimal dynamic programming- (DP-) based exhaustive search method are listed in Table 2. Compared to the optimal solution from DP, the improvement obtained by our method is only 1% worse

Table 2: Results of our method and the optimal DP method. Four sizes are available for each cell. $\%\Delta T$ is the percentage timing improvement, $\%\Delta A$ is the percentage change of total cell area (negative value means deterioration).

Ckt	Number of cells	Number of crit. cells	DNF				Opt. DP			
			Delay (ns)	$\%\Delta T$	$\%\Delta A$	Runtime (sec)	Delay (ns)	$\%\Delta T$	$\%\Delta A$	Run time (sec)
C432	160	50	3.2	11.9	−9.9	9	3.1	13.2	−10.0	103
C499	202	51	3.5	11.9	−9.4	14	3.5	12.4	−9.3	119
C880	383	77	3.5	14.8	−9.9	19	3.4	16.2	−10.0	195
C1355	544	85	4.5	6.8	−8.1	22	4.4	7.0	−8.8	489
C1908	880	88	5.5	16.1	−9.8	39	5.4	17.0	−10.0	556
C2670	1.3 K	91	3.4	9.6	−9.5	38	3.4	11.1	−9.7	908
C3540	1.7 K	124	5.0	16.8	−9.0	69	4.8	19.0	−9.4	1724
C5315	2.3 K	138	5.3	10.0	−6.9	70	5.3	10.2	−7.9	3228
C6288	2.4 K	299	14.1	11.5	−8.1	97	14.1	11.7	−9.1	14998
C7552	3.5 K	199	7.3	9.9	−8.2	112	7.2	11.5	−7.5	6847
Average	1336	120		11.9	−8.9	48		12.9	−9.2	2916

Table 3: Results of our method and the optimal DP method with Synopsys 90 nm library [20].

Ckt	DNF				Opt. DP			
	Delay (ns)	$\%\Delta T$	$\%\Delta A$	Run time (secs)	Delay (ns)	$\%\Delta T$	$\%\Delta A$	Run time (secs)
C432	1.7	8.5	−9.1	10	1.7	11.7	−9.9	92
C499	1.0	12.8	−9.9	15	1.0	13.2	−9.9	144
C880	1.5	10.5	−9.8	17	1.4	14.7	−10.0	148
C1355	1.0	7.0	−8.8	40	1.0	8.1	−9.0	705
C1908	1.6	12.1	−10.0	72	1.5	16.2	−10.0	1024
C2670	1.3	9.4	−10.0	44	1.3	13.1	−10.0	884
C3540	2.2	13.5	−10.0	105	2.1	17.9	−10.0	1650
C5315	1.6	9.9	−9.4	280	1.5	11.8	−9.5	8410
C6288	6.0	14.2	−10.0	312	5.8	18.2	−10.0	25689
C7552	1.7	10.2	−9.8	364	1.7	13.8	−9.8	13204
Average		10.9	−9.7	126		13.9	−9.8	5195

(11.9% versus 12.9%); note that the solutions of both methods satisfy the 10% area increase constraint. Furthermore, our run time is 60X less than that of the exhaustive search method even with all three pruning conditions. Note again that we cannot compare our technique directly to that of [3], since we do not have their cell library or parameters.

We have also tested our algorithm using the more advanced and industry-like 90 nm library from Synopsys [20]. The results are given in Table 3. A similar trend to that of Table 2 is obtained is with the same initial sizing and 10% area relaxation, our method is only 3% worse (10.9% versus 13.9%) than the optimal solution. The run time we use is 40X less than the optimal one on average. We observe that there is a slightly increase in the optimality gap (from 1% to 3%) for 90 nm library compared to the 180 nm library. Our conjecture is that this increase is mainly due to the increased sensitivity in the 90 nm library; that is, for the same amount of cell size change, the relative delay change in the 90 nm library can be about 40% more than that of 180 nm library. Thus small errors in the optimal choice of cell sizes in near-optimal methods such as ours can lead to somewhat larger errors in the circuit delay results for libraries with larger sensitivity. However, we should note that, even with such large sensitivity increase as mentioned above, our optimality gap increases by only 2%.

To show the scalability of our algorithm, two additional experiments were performed with the 180 nm library. In the first one, the sizes of all cells were considered for resizing rather than only cells in $\mathcal{P}_\alpha$. The results are listed in Table 4(a). Compared to only focusing on $\mathcal{P}_\alpha$, the average number of cells that are resizable is increased by 10 times from 120 to 1336, the run time is also increased by about 10 times from 48 secs to 525 secs, while the timing improvement % is an absolute of 2% better. The run time plot with respect to the number of resizable cells is shown in Figure 10, which best fits a linear function. This is much lower than the worst-case complexity we derived in Section 6, and in keeping with the well-known much smaller empirical time complexity of the network simplex method compared to its worst-case complexity [19].

In the second experiment, we expanded the cell library by adding six artificial size options with proportional driving resistances and input capacitances for each cell (three size options are added between w_3 and w_4 with uniform spacing between them, and the other three added options are made

Table 4: (a) Results of our method when all circuit cells are considered for resizing with four sizes for each cell. (b) Results of our method when ten size options available for each cell; only cells in $\mathscr{P}_\alpha$ are resizable.

(a)

ckt	%ΔT	%ΔA	Run time (secs)
C432	12.9	−9.9	95
C499	13.5	−9.7	97
C880	14.9	−9.8	128
C1355	9.4	−9.5	249
C1908	19.9	−9.7	309
C2670	9.7	−9.4	449
C3540	22.6	−9.9	698
C5315	10.0	−6.9	901
C6288	13.2	−8.9	1097
C7552	12.1	−8.5	1229
Average	13.9	−9.2	525

(b)

ckt	%ΔT	%ΔA	Run time (secs)
C432	14.8	−9.7	72
C499	16.6	−9.8	140
C880	14.9	−9.2	144
C1355	13.3	−9.9	208
C1908	19.9	−9.1	344
C2670	12.7	−9.8	315
C3540	23.1	−9.7	527
C5315	10.8	−8.5	476
C6288	15.0	−9.5	698
C7552	14.9	−8.9	1087
Average	15.6	−9.4	401

larger than w_4. The intervals between the last three newly added size options are the same as between the first three newly added size options. We use linear approximation determined from the four options provided in the original library to calculate the driving resistances and input capacitances of the added options), so that each cell has ten different size options. The results with this larger library and cells in $\mathscr{P}_\alpha$ being resizable are shown in Table 4(b). With the larger library, we can only obtain the optimal results for the two smallest circuits using the exhaustive search method. Our results are only 2.5% worse (14.8% versus 17.3%) for circuit C432, and 2.0% worse (16.8% versus 18.8%) for C499 compared to the optimal solutions. The run times of our method are about 80X less than the exhaustive search method for C432 (72 secs versus 5343 secs) and over 135X less for C499 (140 secs versus 18993 secs). Compared to the four-option results in Table 2, our DNF method's run time is increased by about 7 times, while the timing improvement % is increased by an absolute of 3.7%. We also plot in Figure 11 the run time with respect to the number S of available size options for each cell for three representative circuits C432, C1908, and C7552. The plot for C1908 best matches a cubic function of S, and the plots for C432 and C7552 best match quadratic functions, which are all consistent with the upper bound time complexity derivation given in Section 6 (that the run time is proportional to S^4). However, since, in current VLSI circuits, S and even S^4 are generally much smaller than N, the number of cells being re-sized, the dominant run time function is the one shown in Figure 10 that is linear in N.

Finally, we ran our DNF method on the complex ISPD 2012 contest benchmarks [21]. The sizes of the benchmarks range from 25 K to 959 K cells. For each type of cell, there are 30 different "sizes" (the sizes are combinations of actual sizes and different threshold voltages) in the library. For each circuit, the power constraint is determined by power optimizing the fast version of each ISPD benchmark using an internal tool (this tool uses the more complex DNF formulation modeled as a fixed-charge network flow problem that was alluded to in Section 2) that was part of the ISPD'12 competition and was in the top 6 (out of 17 teams) for 6 of the circuits; see [23]. Then, we perform our timing-driven sizing under this leakage power constraint. We run three rounds of our DNF-based sizing. In the first round, all cells in the circuit are sizable; this is needed as the ISPD'12 circuits also come with an initial sizing, and these correspond to very high delays and low power designs. In the second round, we perform further improvement by focusing on sizing only cells on critical and near critical paths (with delay ≥90% of the max path delay); note that this is the only round we do for the ISCAS'85 circuits. In the last round, we perform final adjustment, again using the DNF method, to only sub-paths in the critical paths that show delay reduction potential. The delay reduction potential is measured by the differences in the delay of the size selection s chosen in the second round, and the adjacent sizes of s, and the ratio of the corresponding power increases to the current positive leakage power slack of the circuit.

Both the circuit sizes of the ISPD'12 benchmark and the number of sizing options per cell are far beyond the capabilities of the optimal DP method (it runs out of memory for the smallest circuit). We thus compared our method to a state-of-the-art cell-sizing method in [3] (implemented by us), which also uses an approximate dynamic programming (DP)—it has a nonoptimal similarity-based partial solution pruning that can significantly reduce the number of partial solution generated. The results are shown in Table 5. As we can see, we achieve almost the same delay quality as the DP-based method (only 0.9% above it on the average) but use less than half of its run time. This highlights the efficiency of our method in achieving high quality results. We also show the run time plot with respect to the number of cells in Figure 12. Again, a linear scalability with respect to the number of cells is seen for our DNF method.

9. Conclusions

We presented a novel and efficient timing-driven network flow-based cell-sizing algorithm. We developed a size option selection graph, in which cell size options are modeled as nodes, and the cost of flows passing through various nodes is equal to the change in the timing objective function when the cell sizes corresponding to these nodes are chosen. Thus,

Table 5: Results of our DNF method and the approximate DP method of [3] for the ISPD 2012 cell-sizing benchmark suite. The "% Delay Impr." column shows the percentage difference of the delay between ours and the Approx. DP method (positive value indicates our method is better, negative indicates that the Approx. DP method is better).

Ckt	Number of of cells	Power Con-str. (w)	DNF				Aprrox. DP		
			Delay (ps)	Power (w)	Run time (h)	% Delay Impr.	Delay (ps)	Power (w)	Run time (h)
DMA	25 k	1.2	770	1.2	1.3	−0.7	765	1.2	2.8
pci_br32	33 k	0.9	676	0.8	2.0	−0.9	670	0.9	3.5
des_perf	111 k	7	795	7	7.1	−0.9	788	7	12.4
vga_lcd	165 k	0.7	673	0.7	5.6	0.6	677	0.7	14
b19	219 k	2.3	2126	2.3	10	−2	2084	2.3	19
leon	649 k	2.1	1632	2.1	22.2	−0.9	1616	2.1	48.4
Netcard	959 k	2.5	2218	2.5	27.2	−1.4	2186	2.5	58
Average					10.7	−0.9			22.6

by solving for a mincost flow, we can determine the cell sizes that can optimize the circuit delay. Various techniques are proposed to ensure that we can obtain, from the continuous optimization of standard mincost flow, a valid "discrete" mincost flow that meets the discrete mutual exclusiveness condition of cell-size selection. Area and leakage power constraint satisfaction is also taken care of by special network flow structures. The results show that the timing improvement obtained using our method is near optimal for ISCAS'85 benchmarks and similar to a state-of-the-art method for the ISPD'12 benchmarks (the near optimality could not be determined for these circuits as the optimal DP method ran out of memory for the smallest circuit) while being more than twice as fast as that technique. Furthermore, our technique scales well with problem size since its worst-case time complexity is polynomially bounded, and the empirical time complexity is linear.

Acknowledgment

This work was supported by NSF Grants CCF-0811855 and CCR-0204097.

References

[1] C. P. Chen, C. C. N. Chu, and D. F. Wong, "Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 7, pp. 1014–1025, 1999.

[2] J. Fishburn and A. Dunlop, "Tilos: a posynomial programming approach to transistor sizing," in *Proceedings of International Conference on Computer-Aided Design*, pp. 326–328, 1985.

[3] S. Hu, M. Ketkar, and J. Hu, "Gate sizing for cell library-based designs," in *Proceedings of the 44th ACM/IEEE Design Automation Conference (DAC '07)*, pp. 847–852, June 2007.

[4] F. Beeftink, P. Kudva, D. Kung, and L. Stok, "Gate-size selection for standard cell libraries," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '98)*, pp. 545–550, November 1998.

[5] O. Coudert, "Gate sizing for constrained delay/power/area optimization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 5, no. 4, pp. 465–472, 1997.

[6] M. M. Ozdal, S. Burns, and J. Hu, "Gate sizing and device technology selection algorithms for high-performance industrial designs," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '11)*, pp. 724–731, November 2011.

[7] S. Dutt and H. Ren, "Discretized network flow techniques for timing and wire-length driven incremental placement with white-space satisfaction," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 7, pp. 1277–1290, 2011.

[8] H. Ren and S. Dutt, "A provably high-probability white-space satisfaction algorithm with good performance for standard-cell detailed placement," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 7, pp. 1291–1304, 2011.

[9] S. Dutt, H. Ren, F. Yuan, and V. Suthar, "A network-flow approach to timing-driven incremental placement for ASICs," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD '06)*, pp. 375–382, November 2006.

[10] U. Brenner, "VLSI legalization with minimum perturbation by iterative augmentation," in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE '12)*, pp. 1385–1390, March 2012.

[11] H. Ren and S. Dutt, "A network-flow based cell sizing algorithm," in *Proceedings of the 17th International Workshop on Logic & Synthesis*, pp. 7–14, 2008.

[12] S. Dutt and H. Ren, "Timing yield optimization via discrete gate sizing using globally-informed delay PDFs," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '10)*, pp. 570–577, November 2010.

[13] H. Ren and S. Dutt, "Effective power optimization under timing and voltage-island constraints via simultaneous VDD, Vth assignments, gate sizing, and placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 5, pp. 746–759, 2011.

[14] H. Ren and S. Dutt, "Algorithms for simultaneous consideration of multiple physical synthesis transforms for timing closure," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD '08)*, November 2008.

[15] A. Nahapetyan and P. M. Pardalos, "A bilinear relaxation based algorithm for concave piecewise linear network flow problems," *Journal of Industrial and Management Optimization*, vol. 3, no. 1, pp. 71–85, 2007.

[16] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, chapter 10-11, Prentice-Hall, Upper Saddle River, NJ, USA, 1993.

[17] D. Kim and P. Pardalos, "Gate sizing in MOS digital circuits with linear programming," in *Proceedings of the Conference on European Design Automation (EURO-DAC '90)*, pp. 217–221, 1990.

[18] R. K. Ahuja and J. B. Orlin, "Scaling network simplex algorithm," *Operations Research*, vol. 40, supplement 1, pp. S5–S13, 1992.

[19] I. Adler and N. Megiddo, "A simplex algorithm whose average number of steps is bounded between two quadratic functions of the smaller dimension," *Journal of the ACM*, vol. 32, no. 4, pp. 871–895, 1985.

[20] Synopsys 90 *nm* Library, http://www.synopsys.com/community/universityprogram/pages/library.aspx.

[21] ISPD 2012 cell sizing contest, http://www.ispd.cc/contests/12/ispd2012_contest.html.

[22] X. Yang, B. K. Choi, and M. Sarrafzadeh, "Timing-driven placement using design hierarchy guided constraint generation," in *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD '02)*, pp. 177–180, November 2002.

[23] http://www.ispd.cc/contests/12/ISPD_2012_Contest_Results.pdf.

Hindawi Publishing Corporation
VLSI Design
Volume 2012, Article ID 312808, 11 pages
doi:10.1155/2012/312808

Research Article

A Graph-Based Approach to Optimal Scan Chain Stitching Using RTL Design Descriptions

Lilia Zaourar,[1] Yann Kieffer,[2] and Chouki Aktouf[3]

[1] *SOC Department, LIP6 Laboratory, University Pierre and Marie Curie, 4 Place Jussieu, 75252 Paris Cedex 05, France*
[2] *LCIS, Grenoble Institute of Technology and University of Grenoble, 50 Rue Barthélémy de Laffémas, 26000 Valence Cedex, France*
[3] *DeFacTo Technologies, 167 Rue de Mayoussard, 38430 Moirans, France*

Correspondence should be addressed to Lilia Zaourar, lilia.zaourar@lip6.fr

Received 30 April 2012; Accepted 6 November 2012

Academic Editor: Shantanu Dutt

Copyright © 2012 Lilia Zaourar et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The scan chain insertion problem is one of the mandatory logic insertion design tasks. The scanning of designs is a very efficient way of improving their testability. But it does impact size and performance, depending on the stitching ordering of the scan chain. In this paper, we propose a graph-based approach to a stitching algorithm for automatic and optimal scan chain insertion at the RTL. Our method is divided into two main steps. The first one builds graph models for inferring logical proximity information from the design, and then the second one uses classic approximation algorithms for the traveling salesman problem to determine the best scan-stitching ordering. We show how this algorithm allows the decrease of the cost of both scan analysis and implementation, by measuring total wirelength on placed and routed benchmark designs, both academic and industrial.

1. Introduction

The design flow of an integrated circuit (IC), meaning the software applications that allows the designer to move from its specification to its concrete realization, involves many stages of optimization problems, usually from system level to layout [1–3]. Indeed, the realization of an IC is a costly process both in time and money, and requires large engineering resources. In addition to technological advances, which continuously improve the efficiency of chip manufacturing techniques, as the fine prints on silicon for example, the continuous increase of the power of computers offers great opportunities for improving the process of designing and manufacturing complex ICs.

With both recent advances of the semiconductor industry and new market constraints, Time To Market (TTM) and product quality are becoming major issues. The circuit must meet flawlessly customer expectations in terms of functionality, speed, quality, reliability, and cost. In such a challenging economic environment, and given the significant level of complexity which is reached by the IC, manufacturing testing is more than ever an important factor in the design problem. Today, chip testing should be short, efficient, and cost-effective. A significant amount of research work is ongoing with a focus on complex design-for-test problems at both universities and industry. Design For Test (DFT) techniques are becoming a key since they are considered during the chip design process and flow. Cost of manufacturing chips averages the cost of testing them. So the semiconductors community needs low cost and high quality test solutions. New and efficient DFT solutions are greeted with higher expectations than ever. Most current DFT solutions result in unpredictable design development time and development costs and directly impact (TTM).

In this context, DFT tools are receiving growing attention with the advent of core based System On Chip (SOC) design. In particular, when cores from different vendors are integrated together on the system on chip (as it is often done nowadays), the difficulty level of testing grows rapidly. Among the most apparent issues are core access, system diagnosis, test reuse, test compaction, tester qualification and Intellectual Property (IP) protection. The ITRS (International Technical Roadmap for Semiconductor) identifies key technological challenges and needs facing the semiconductor

industry through the end of the next decade. Difficult near-term and long-term testing and test equipment challenges were reported in [4].

Several of the DFT solutions for ASIC and ASIP designs are based on the internal Scan DFT technique. Scan is potentially an efficient technique, if used properly. Currently, full Scan is the most widely used structured DFT approach where all the design sequential elements belong to the scan architecture or scan chains [5]. A Scan-based DFT architecture provides the ability to shift information by scanning the set of states of the circuit. All flip-flops are chained to each other to allow the introduction of test vectors as input and retrieval of test results as output. Thus, the resulting shift register is fully controllable and observable through the primary inputs and outputs of the circuit. This makes Scan testing widely adopted for manufacturing testing and other purposes such as silicon debug. However, we do care about hardware overhead (area, pin count, and so on), performance penalty, and extra design effort that may be associated with full scan insertion during the chip design process. Therefore, the primary objective in DFT is to automate the insertion of the test logic and to minimize the impact of test circuitry on chip performance and cost. Thus it is important and essential to optimize the scan chain insertion process.

However, traditional Scan solutions make such an extension very difficult since engineers need to handle gate-level netlist without taking benefit from what happens during synthesis optimization. Also, Scan implementation decisions are considered post synthesis. This is too late in comparison to RTL design decisions. Adopting Scan at the Register Transfer Level (RTL) will cover new design and manufacturing needs, strengthen verification, and consolidate reusable design methodologies by closing the gap between RTL and design for test. There are many advantages to inserting scan at the RTL level like: benefiting from the synthesis process (i.e., better optimization in terms of area and timing), the ability to debug testability issues early in the design flow, and leveraging the optimization done by the synthesis tool. The possibility to insert scan at the RTL dates back to the late nineties [6–8]. Although this idea did not get a widespread attention in the meantime, an EDA tool that lets one do it, namely HiDFT-SIGNOFF by DeFacTo Technologies, has appeared and is commercially available. Its main use today is to help debug testability issues early on in the design flow, thus helping avoid costly iterations around the synthesis step. To achieve this purpose, Scan chains are introduced at the RTL, bringing to light any problems in doing so—like noncontrollable Flip-Flops (FFs), for example. Then the scanned design can be used in other estimates, but will be discarded as far as the main flow is concerned: the real scan chains that will end up being implemented in the chip are still inserted at gate-level, using a traditional design flow. Before one can insert the actual scan chains at the RTL, one big hurdle has to be overcome. The impact of scan insertion on the design cost and performance can be important if the ordering of the FFs in the scan chain is not picked carefully. But the information allowing for such a choice, namely placement information of the FFs, is only available at the back-end of the design flow, far away from the point where the RTL code for scan must be finalized. Our goal in this work is to develop a tool to automatically generate optimal scan chains at RTL in terms of area and additional test time for a given circuit, while respecting a number of electronics constraints.

To analyze the optimal design location where full scan chains need to be inserted, the following considerations are required. First, to implement a scan chain, one has to have knowledge of the Flip-Flops (FFs) of the design. Second, the best place to insert any new task in a flow is at the earliest. The motivations for an early scan insertion are threefold: the complexity of the objects grow as one goes forward in a flow, so data handling gets more costly; whatever is added to the design is better if integration happens earlier in the flow; and finally, should the treatment lead to iterations—either redesigning or iterations in the flow—the sooner the iterations, the lower the cost. Third, at the point where insertion is finalized, all the information required for the insertion has to be available. If this information depends at least partly on the insertion itself, this leads to iterations—see below. Fourth, insertion can start at some level in the flow, and end at a later point in same. Again, this can be undesirable, but also unavoidable once some design decisions on the insertion process have been taken. Since FFs are usually known after synthesis, once a gatel-level netlist is available, this sounds like a reasonable point in the flow to insert scan. But the replacement of normal FFs by scan FFs has to be done before synthesis ends, since timing closure is affected by it. We will see in the next section that FFs detection can happen before synthesis.

On the other hand, we will argue in Section 3 that the scan stitching ordering is an important part of making scan insertion seamless. To compute an optimized ordering, the best information to have is that of the placement of the FFs in the layout. If one uses placement information, scan insertion should be finalized during placement and routing (at the earliest). This is the classical scheme for scan insertion, and it is this one that is used in industrial EDA tools for scan insertion. This is why all scan insertion tools today either provide a costly ordering, or must decide the ordering in several phases. The next paragraph reviews the state of the art, while keeping an eye on these different criteria.

The traditional way to insert full scan in a design is to replace the FFs by scanned FFs during synthesis; possibly connect them into a chain; and then during place and route iterations, (re)connect them to try to minimize the total Manhattan length of the added connections. The exact place where one should reoptimize the ordering of the chain is a matter of debate, and heavily depends on the flow used. For an in-depth discussion of this topic in the case of the Synopsys flow, see [9].

Also, scan implementation decisions are considered post synthesis. This is too late in comparison to RTL design decisions. Adopting scan at the RTL level will cover new design and manufacturing needs, strengthen verification and consolidate reusable design methodologies by closing the gap between RTL and design for test. There are many advantages to inserting scan at the RTL level like benefiting from the

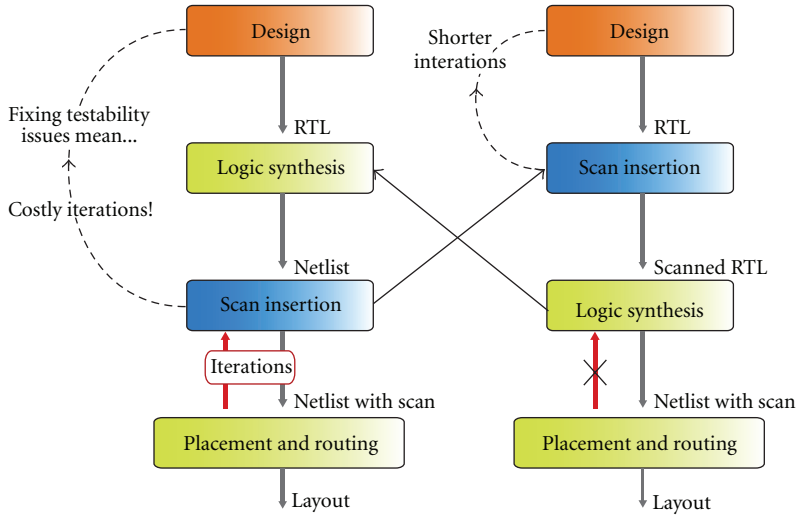


Figure 1: Classical scan insertion versus our method.

synthesis process (i.e., better optimization in terms of area and timing).

Most of the work done on scan chains insertion and stitching ordering optimization assumes that placement information is available [9–14], meaning that it is actually reordering optimization. In all the cases, scan (re)ordering is reduced to the problem of the Traveling Salesman Problem in a suitable graph. An additional assumption is added in [15]. The author proposes a procedure to find and break intersections. This is also done to reduce the overal wire length.

The knowledge of FFs is necessary to implement a scan chain. It has been suggested several times independently that one does not need a netlist to have knowledge of FFs of the design; this is the basis of higher-level scan insertion [6–8, 16, 17]. Although fault models are defined in terms of gates and nets, scan insertion itself can actually be done at the RTL. This is achieved by describing the behaviour of the scan chain directly at the RTL. It will then be translated by the synthesis tool into added nets between FFs, and multiplexers in front of FFs. No scanned FFs are used in this case; we give a more detailed account in the Section 3. It would seem that inserting scan at the RTL makes the problem of scan stitching reoptimization much worse. This may be one reason why in that case, the reoptimization possibility is commonly dropped, in favour of a single optimization at the time of insertion.

Most studies on higher-level scan either do not mention optimization [7], or mention local optimizations only, meaning trying to reuse bits of functional paths for the scan path, thus reducing the need for added nets and additional multiplexing logic [6, 16–18]. In [8] an attempt is made to achieve both local and global optimizations by carefully delineating a graphic model for the scan ordering problem. This graphic model rests on an analysis of the RTL source code. Unfortunately, the algorithm they propose was never implemented [19], and thus could be tested only on small designs.

In this paper, we try to close the gap between RTL Scan insertion and actual scan stitching optimization, while retaining the model of one-time insertion without any later reoptimization see Figure 1.

More specifically, we tackle the following optimization problem: given an RTL description of the design (Verilog or VHDL) we have to find a stitching ordering of the memory elements in the scan chain, which minimizes the impact of test circuitry on chip features (area, power) while keeping testing time at its minimum.

The remaining of this paper is organized as follows. Section 2 presents how our approach to RT-level scan is novel. Section 3 describes the problem of scan chain insertion at the RTL. It explains how RTL scan is implemented and gives arguments as to why wirelength is an efficient global measure of the scan insertion cost. In Section 4, we propose our RTL scan stitching algorithm together with the graph models on which it is based. Implementation, experimental results on both academic and industrial designs are given in Section 5, followed by discussions that RTL scan is a viable alternative to gate-level scan. Finally, Section 6 concludes the paper and points out some research directions for future work.

2. Original Algorithmic Content of This Work

To the best of our knowledge, this work is the first to offer a formal treatment of the scan stitching ordering problem as a discrete optimization problem in the case of RT-level scan insertion. We formalize the problem; give reductions from the several-chains problem to the one-chain problem; and solve the one-chain problem in two steps. The first step, which can be seen as a kind of preprocessing, allows us to build a graph representative of the actual optimization. Then in a second step, algorithms for the TSP are adapted to actually build the chain.

A part of this work has been presented before in [20] and recenlty in [21, 22]. The present paper is aimed at

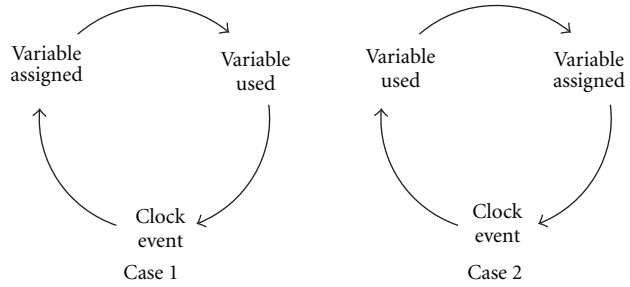


Figure 2: Two use cases for variables in VHDL.

giving a more complete presentation for both theoretical and experimental contributions. In fact, we give here all of the theoretical basis and algorithmic elements of this work including detailed implementation that has not been presented before. Also some numerical results and comparisons resulting are added.

The main contributions of the paper are summarized below:

(i) we analyze what is a suitable objective for measuring the quality of scan stitching orderings,

(ii) we give a mathematical formulation of the problem of scan insertion at the register transfer level (RTL),

(iii) we give two reduction procedures to solve the several-chains variants based on a routine solving the one-chain case,

(iv) we solve the one-chain case in a two-steps approach,

(v) we evaluate our algorithm on both academic and industrial designs.

Our algorithm provides a basis for considering the implementation of scan chains as soon as the RTL of a block is available; the authors think that the lack of optimization has been a big obstacle to its adoption in design flows. The methodology has been validated by our industrial partner DeFacTo Technologies in the tool HiDFT-SIGNOFF, it is a first step towards considering the integration of scan at the RTL.

3. Scan Insertion at the RT-Level

We illustrate Scan insertion at RTL with the VHDL language. Scan insertion is a three steps process: first identify which signals and variables will give rise to flip-flops in the netlist; second, decide which ordering will be used to chain the FFs together; third, change the RTL code to offer a (new) testing mode for the design.

Since the object of this work is the second step of this whole process, we now restrict our attention to its first and third steps. This section is included mainly to make this article self-contained; for a more thorough presentation, the reader is referred to [6–8].

We first present how flip-flops are identified; then we illustrate RTL edition for introducing scan at RTL; finally, we examine what could be a good measure of the impact of scan insertion on a design.

```
process(clock)
begin
   if clock' event and clock='1' then
      Z <= X or Y;
      Y <= X and Y;
   end if;
end process;
```

Algorithm 1: Example VHDL process.

```
process(clock)
begin
   if clock' event and clock='1' then
      if scan_en='1' then
         Y <= X;
         Z <= Y;
      else
         Z <= X or Y;
         Y <= X and Y;
      end if;
   end if;
end process;
```

Algorithm 2: VHDL process enriched with scan code.

3.1. Flip-Flops Identification. FFs identification is realized process by process. Each process is searched for variables and signals. All signals in a process will be translated into a FF in the netlist. To determine which variables will give rise to FFs, one has to first identify clocking signals. Then two cases can happen. To determine which applies in a particular case, one has to recall that processes execute cyclically—see Figure 2.

Either the variable is assigned between the clocking event in the process and the point where the variable is used (case 1). In that case, no FF is generated. Or the clocking event lies between the point where the variable is assigned and that where the variable is used (case 2). In that case, memorization has to occur, and a FF is generated.

3.2. Test-Mode Introduction by RTL Edition. To illustrate the introduction of a scan chain at RTL through RTL code edition, we consider the simple process in Algorithm 1.

In order to introduce a testing mode behavior to the design, we simply describe in VHDL what the process is going to do in test mode. The additional VHDL code is shown in Algorithm 2 in boldface; it consists of a conditional statement on the value of the scan_en signal, and assignments expressing the functional chaining of the registers of the process. Note that this (too simple) example is misleading in showing a doubling of the size of the RTL code; code growth for realistic designs is much less than that.

3.3. Wirelength as the Prime Parameter for Stitching Ordering. In most of the literature about scan ordering optimization, wirelength is the objective used to guide the optimization process. In these works, placement of the FFs is known; so wirelength is a quantity that is directly available during

optimization. We will give our own argument for considering wirelength the most useful parameter to optimize stitching orderings.

When adding logic to a design for non-functional reasons, one tries to minimize the increase in size of the design. Additional area due to scan comes in two parts. First, FFs have to be instrumented either into scanned FFs, or with the help of a multiplexer at their input. In the former case, the area cost depends only on the number of FFs in the design; in the latter, the area cost can be less than the maximum if some optimization takes place during synthesis. In both cases, that cost is bounded independently from the stitching ordering. Second, wires have to be added between the output net of a FF and the next FF in the scan chain—either the scanned FF, or one input of the multiplexer in front of it. This does not represent an area cost in itself, since routing happens in the upper metallic layers. But it does add to the difficulty of placement and routing, with the possibility to have a very degraded situation if the stitching order is not chosen with care. In that case, the area can grow if routing is not possible anymore with the available space. We consider this a much more important factor than the additional cell area; therefore our measure of ordering quality will be based on it.

It is not possible to express the impact of added wires on placement and routing as a simple function of these wires; the impact will depend also on the sparsity of the design. But since we take a worst-case approach to the impact of wires, we will use the added wirelength due to scan insertion as our optimization function.

This choice will be validated by the variability that is observed on this parameter—see Section 5.

We now review quickly all the other parameters scan insertion could have an impact on.

In order to help ensuring timing closure, the maximum of the length of the added wires would be a reasonable measure. But since in our case synthesis happens after scan insertion, it will be up to the synthesis-and later on, placement and routing-tools, to ensure timing closure, using all the flexibility of gate sizing to achieve it. Note also that ensuring a low total added wirelength means that not too many added wires will be long, hence lowering the impact on timing, and the additional load for the synthesis tool.

Power while in functional mode grows with wirelength; so minimizing wirelength will also lower the impact on functional power. Power while in test mode is dominated by switching: the values shifted through the scan chain force the values in FFs to change more often than they would in functional mode, leading to power supply issues. A common remedy to this condition is to take benefit of don't-care values in test vectors to fill them in a way to minimize switching. This also works with RTL-scan. There have been many attempts in other works at minimizing power during test mode by changing the stitching ordering. We have not followed this trail; it would be worthwhile to try combining it with our approach based on minimizing wirelength.

Finally, observability and controllability are not impacted by the stitching ordering. We note here that although combinatorial parts of the circuit have the same observability and controllability in a full scan design implemented both at gate and at RT-level, in the case of RT-level scan, there will be more stuck-at faults, since the multiplexers are now no more combined with FFs. Hence fault coverage values tend to differ, although not much, between both methods for scan insertion.

Having established that wirelength should be minimized, we now turn to the precise description of the optimization problem we will consider in order to try to find low wirelength stitching orderings.

4. Stitching Algorithm for Optimal Scan Chain Insertion at the RT Level

To determine the scan chains that best meets the needs of different integrated circuit designers, while maintaining maximum constraints and restrictions related to the electronic problem, we propose a new scan stitiching algorithm for the automatic insertion of optimal scan chains at the RTL. Our algorithm is structured into two phases, as shown in Figure 3. The first phase aims to build a graph-based model to translate the problem and its constraints from an RTL description to a generic mathematical model, in the form of an undirected graph. In a second phase, scan chains are determined through computations in that graph. The stages in each phase are outlined in the next paragraphs.

4.1. Phase 1: Graph Extraction. The challenge here is to be able to take into consideration still at high level what is going to happen later on during the placement and routing (P&R) steps. To build this model, we extract from the RTL description information on the expected proximity of the memory elements in the layout.

This phase is devised into three steps, as follows.

4.1.1. Design Elaboration (Lightweight Synthesis). First, we perform a preprocessing called Elaboration. Although scan logic can be described at RTL, the very elements of which fault models are talking—nets, gates and FFs—do not exist yet at this level. Hence we need first to translate the RTL code into these elements. This is what synthesis does. But it is not feasible to have a full synthesis at this step in the flow. Once RTL-scan is inserted, the synthesis step still has to be done; we do not want to duplicate that effort.

Our solution relies on a lightweight synthesis as the first step of the RTL scan insertion. This synthesis is done in terms of a virtual library of generic (non-physical) gates; it does not try to optimize logic, timing or gate sizes; it does not need the user to do any fine-tuning; in short, it is done transparently. The user of the scan insertion tool only provides his RTL code, and will get back a scanned RTL code: he will never see the netlist that comes out of the lightweight synthesis. Indeed, in our method, this netlist serves only one purpose, namely graph extraction.

4.1.2. Building the D-Graph. The second step after the design elaboration is to build the undirected D-graph (for Design graph), denoted by $G_D = (V_d, E_d)$. It represents the RTL description of the design. It is constructed as follows (cf.

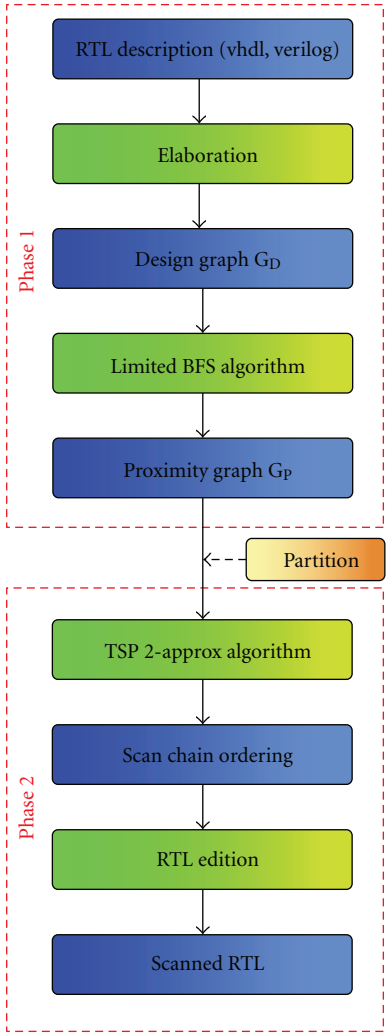


Figure 3: Stitching algorithm for optimal scan chain at RTL.

Figure 4). Each memory element, gate and net from the elaborated netlist corresponds to a vertex (V_D) of the graph G_D. There is an edge (E_D) between two vertices if and only if the corresponding elements in the RTL description or in the netlist are connected.

Note that the vertices are partitioned into two sets: nets on one hand, and gates and memory elements on the other; every edge in the graph has one end on each side. Therefore, G_D is a bipartite graph.

The graph G_D is undirected because it only aims at giving indications on how close memory elements might be located at the placement step; the direction of information flow along the nets is of no consequence for the decision in our algorithm to stitch together two FFs.

4.1.3. Building the P-Graph for Memory Elements Proximity. The third step is to extract information from the design that is necessary for scan chain stitching optimization (second phase). This information will be given in the form of an edge-valuated Proximity Graph (in short *P*-graph), to which we will now refer to as G_P.

The *P*-graph G_P will be built from the graph G_D as follows: the vertices (V_P) are the memory elements of the design; there is an edge (E_P) between two vertices if there is direct electronic connection in the design between the memory elements they represent; this connection is not allowed to go through other FFs. Additionally, the edges are labeled by the length of a shortest non-oriented path between the corresponding vertices in the graph G_D. The weight on edge ij will be denoted w_{ij}. It represents the Path lengths between the pair of memrory elements that it matches.

Path lengths are restricted to a threshold value t which is a fixed parameter of our algorithm: for a longer (shortest) path between two memory elements, no edge is put in G_P.

To compute G_P, we use a variant of the algorithm of Breadth First Search (BFS) [23] that limits the depth of the exploration to find the limited shortest path. This algorithm is called Limited-BFS in the following. The graph computed with the help of Limited-BFS is used to estimate a probable geometric proximity of memory elements in the Layout.

Figure 4 illustrate the construction of G_D and G_P on an example design.

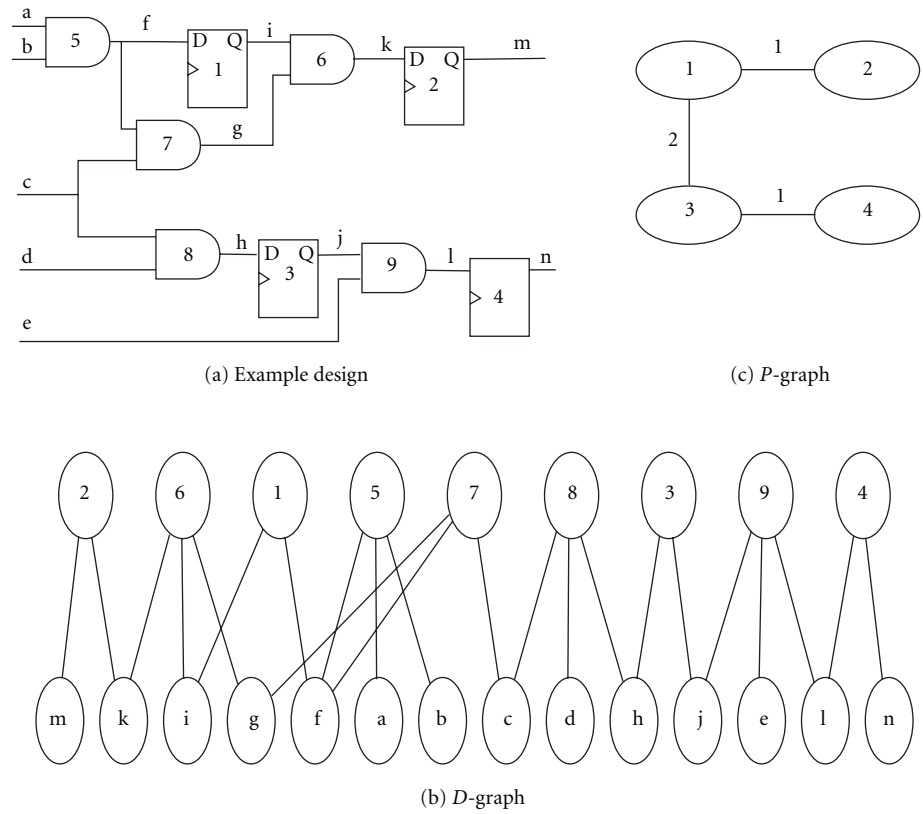
4.2. Phase 2: Construction of Scan Chains. Using this new formalism, the chaining of memory elements in the circuit corresponds to a partition of the vertices of G_P (representing the FFs of the design) into sequences of vertices, each sequence representing one scan chain. An important constraint in scan chaining asks for all chains to have the same number of FFs. Also, the sequences should preferably use edges from the graph G_P, although adding connections is a possibility. The cost of going from v_i to v_j in the sequence is counted as the length of a shortest path from v_i to v_j in the graph G_P; hence it will be w_{ij} if the edge ij is present in G_P.

In the case of a single scan chain, our problem reduces to the Traveling Salesman Problem [23]. Still, to apply TSP algorithms, the whole cost matrix for each pair of vertices of the graphs has to be precomputed; this is not feasible for designs with more than a couple tens of thousands FFs.

Also, single scan chains are not an option for big designs, where testing time would be prohibitive if scanning were not done using more than one chain.

Before explainning the second phase, we present two algorithmic devices to reduce the general chaining problem to the single-chain case. Then we show how our problem can be reduced to the Traveling Salesman Problem and some algorithms to solve it. Finally, we describe the two steps of the second phase.

Chain Splitting. The simplest way of reducing the several-chains scanning problem to its single-chain subcase is by appropriate post-processing. Once one has computed a single scan chain for the whole design, this chain can be split into the desired number of segments (the actual scan chains). This device is really fast, and it is not expected that the quality of the output of the whole process will be much degraged as compared to that of the single-chaining algorithm. Also, it is very easy to implement, and is our recommandation for the cases where fast enough single-chaining algorithms are available.



(a) Example design

(c) *P*-graph

(b) *D*-graph

FIGURE 4: Example design and associated graphs G_P and G_D.

Graph Partitioning. A more elaborate device is the preprocessing of the graph G_P with the help of a graph partitioning algorithm. If s scan chains are called for, the graph G_P should be partitioned into s parts having equal or similar numbers of vertices. Then finding scan chains for the whole design reduces to finding a single chain for each part of G_P.

Even if the partitioning is handled by an appropriate algorithmic package, implementing this solution is not as easy as chain splitting. But using it brings with it another benefit: the TSP problems to solve in this case are restricted to the length of the scan chains. For test application reasons, these are in actuality limited; although technically feasible, one seldom meets scan chains of 10000 FFs.

Hence this solution helps ease up the problem of computing the costs matrix for the input of the TSP, which is actually the longest step of our methodology.

Another benefit is that when using partitioning, the whole method scales with only a linear increase in running times if the maximum size of scan chains is kept constant.

TSP Algorithms. We now discuss how TSP algorithms can be applied for the single-chain stitching problem.

There are two distinct frameworks for applying TSP algorithms to our problem. The first one allows any algorithm to be used, but it puts constraints onto the size of input graphs that can be fed to it.

The second one is the particular case of an algorithm that can be used directly on G_P, without having to compute costs for edges that are not present in G_P.

Using a Generic TSP Algorithm. The input of a TSP routine is a symmetric matrix representing the costs for edges of a complete graph. In order to apply a TSP algorithm to solve the chaining problem, one needs first to compute the whole costs matrix (except diagonal elements). In our model, we attribute (as cost) to a pair ij that is not an edge of G_P the length of a shortest path between i and j in G_P. The post-processing is very simple: just turn the hamiltonian cycle returned by the TSP routine into a hamiltonian path by removing any edge.

It is the preprocessing step that imposes a serious limitation on the possibility to use this scheme. Indeed, the G_P graph obtained even with small values of the threshold parameter for the Limited-BFS step tends to be quite dense; hence any all-pairs shortest path algorithm puts limit to the input size either through space or time complexity, or both. In practice, it may be feasible to handle 5000 FFs or 10000 FFs blocks, but it is difficult to go much higher and keep a reasonable computation time for design scanning.

If only designs smaller than this are to be treated, then this method is definitely worth trying, all the more that one has then the ability to test and compare different TSP algorithms.

4.2.1. Scan Chain Ordering Using the 2-Approximation for the TSP. A famous textbook algorithm for the TSP, and also a typical example of an approximation algorithm with guaranteed quality, the 2-approximation for the TSP also has nice properties when used on the scan chaining problem.

An algorithm for a minimization problem is called an α-approximation when the solution returned by the algorithm is guaranteed to be no more than α times higher than the optimal solution. Usually, such algorithms presented in the literature are polynomial-time algorithms for NP-hard problems: since the requirement to find an optimal solution is dropped, polynomiality can be recovered.

It may seem unintuitive that one can prove a quality bound without even knowing the value of the optimal solution. Actually, such a proof can be derived through the use of a lower bound on the value of the optimal solutions, through showing that the solution given is less than α times this lower bound.

In the case of the TSP, two approximation algorithms are found in every textbook on the subject: one has an approximation factor of 2, (we call it "the 2-approximation for the TSP"), and the other is Christofides' algorithm, with an approximation factor of 1.5. Both use in their proof, as a lower bound on the optimal tour length, the value of a minimum spanning tree of the graph.

Please note that these approximation ratios are only proven theoretical bounds, and are not enough to compare the empirical behavior of these algorithms. Still, Christofides is bound to give solutions that are not more than 50% away from the optimal one; this would entice one to use this algorithm.

Alas, Christofides uses Weighted Perfect Matching in bipartite graphs as a subroutine, which has to impacts on its use. First, weights have to be precomputed, and we stumble against the same blocks as mentioned in the previous section. Second, Weighted Perfect Matching needs $O(n^3)$ operations, seriously limitting the size of its input.

Without any regret, we turn now to the 2-approximation algorithm, which is seldom considered a useful practical choice because of its loose guaranty, and not-so-good performance on common benches.

The algorithm is in two steps. First, a minimum spanning tree is computed; then a root is chosen, and the tree is explored and vertices ouput with postfix ordering.

Only the first step actually looks at the input graph; and its only requirement is that the graph be connected. If G_P is disconnected, we reconnect it by adding arbitrarily edges between connected components until it is connected.

Thus, using the 2-approximation, we bypass the precomputation of the cost matrix. This means that bigger input graphs can be considered if this algorithm is used, as compared to other TSP algorithms.

Although this algorithm was meant for the TSP, that is for an input being a complete graph with values on all edges, one can prove that the 2-approximation guarantee still holds when the algorithm is applied to a connected graph. In this case, the tour cost has to be understood as the sum of the cost of the edges, where edge costs for inexistent edges are length of shortest paths in the input graphs—hence our choice to model the cost of inexistent edges in this way.

4.2.2. RTL Edition. The final step is the edition of the RTL code of the design. In this step, we insert in the original RTL code/description the additional RTL constructions required to implement the scan testing logic.

5. Implementation and Experimental Results

5.1. Implementation. The method we propose to optimize the stitching ordering of RTL scan chains has been implemented in an experimental version of HiDFT-SIGNOFF. This tool relies on a commercial software library for the parsing and elaboration of the design. This elaboration step is what we called "lightweight synthesis" in Section 4.1.1. The generic library of gates used for the elaboration is the one that comes with the software library.

HiDFT-SIGNOFF has the ability to stitch FFs into several scan chains. This possibility is important from the practical point of view: making several scan chains is the easiest way to reduce test application time. HiDFT-SIGNOFF implements the two strategies presented in Section 4 for handling several chains. One is to stitch a unique scan chain, and then split it into even parts. The other is to first partition the design, and then to stitch a scan chain in each part.

For graph partitioning, the METIS library [24] is used. METIS is one of the most widely used libraries for graph partitioning. It includes two strategies P_{MET} and K_{MET}. P_{MET} uses a multilevel recursive algorithm. The second one, K_{MET}, is implemented with an algorithm of K-partition such that the number of vertices per partition is equal. We use here the second one to partition the graph G_p into K parts.

5.2. Experimental Setup. In order to validate our method, we did experiments on a number of designs, both in VHDL and Verilog. We used the benchmarks 99 ITC [25] which are reference designs for testing integrated circuits. Indeed, these designs have been used extensively in the literature for integrated circuits testing. For each design, a complete description of the circuit with its value is presented on the website. We also used 3 opencore designs [26]: Simple_spi which is a SPI core, Biquad which is a DSP core, and Ac-97, a controller core. All 3 are written in Verilog. These data sets were also used many times in articles on Design and Test.

Table 1 describes these benchmarks. The first line gives the name of the circuit (b09,..., b19) for ITC 99 and three OpenCore (single-spi, biquad, Ac-97). The following lines give the number of memory elements number of FF and the RTL description language (VHDL or Verilog) for each design.

Since our optimization criterion is wirelength, and since we consider congestion during routing an important issue, experimentations were conducted till the place and route step. Two flows were considered, according to Figure 1. In both cases, synthesis, placement and routing were done by Magma's Talus integrated flow. In the case of the gate-level scan insertion flow, the scan chain was also inserted by Talus. No effort options were set for gate-level scan insertion, since

Table 1: Designs description.

Design	No. of FF	Language
b09	28	vhdl
b10	17	vhdl
b11	31	vhdl
b12	121	vhdl
b13	53	vhdl
b14	245	vhdl
b15	449	vhdl
b17	1415	vhdl
b18	3320	vhdl
b19	6642	vhdl
Simple-Spi (SS)	132	verilog
Biquad	204	verilog
Ac-97	2289	verilog

Talus does not offer any. Besides those flows, the designs scanned at the RT level were also fed into Tetramax, in order to check fault coverage rates.

5.3. Numerical Results. Table 2 presents results of the first phase (extraction graphs $G_D = (V_D, E_D)$ and $G_p = (V_p, E_p)$). The columns 1 and 2 give the name of circuits tested (*Design*) and the size in number of memory elements (no. of FF). The following columns give the size of the design graph G_D respectively the number of vertices $|V_D|$ (column 3) and the number of edges $|E_D|$ (column 4). Columns 5 and 6 present the parameters of the proximity graph G_P with the number of vertices $|V_P|$ and the number of edges $|E_P|$. We observe that the number of vertices $|V_P|$ is equal to the number of memory elements of the design (column 2). The last column gives the CPU time for the extraction of the graph G_P from the graph G_D (Times).

The computation of the graph G_P from G_D (cf. Section 4.1.3) involves the threshold parameter t. The practical value of t has been determined empirically. It has to be big enough in order not to put arbitrary limitations on the choices of the algorithm; and it has to be small enough for the design-graph not to fill the whole memory of the computer. For all designs that we tested, the value $t = 4$ was a suitable choice according to those two criteria. This value was used in all our experiments.

The figures for both wirelength after place and route, and computation times, have been gathered in Table 2. The column number of SC gives the number of scan chains for each design. In the wirelength column, GLS denotes the flow with gate-level scan; RTL-scan gives the values for the RTL-scan flow. The slack column gives the ratio (in percent) by which RTL-scan wirelength exceeds that for gate-level scan.

Computation times are given only for the two steps of the stitching optimization method, discarding the time for parsing and elaboration. The column G_P gives the time for constructing the proximity-graph; TSP is the time for the 2-approximation algorithm for the TSP.

We base our analysis of the comparison of wirelength between gate-level and RTL scan on the slack column of Table 3, since the absolute values cannot be directly interpreted. The slack varies between −23% (meaning RTL-scan wirelength is shorter) and 6%. A closer analysis shows that gate-level scan is better mainly for small designs; our method is always better for the bigger designs, with only negative values of slack for all designs with more than 300 FFs.

Table 4 gives the fault coverage obtained by Tetramax for the designs scanned at the RT-level. Let us note that all values are above 98%, and only two lay below 99.8%. We conclude that RTL scan insertion cannot degrade fault coverage too severely as compared to traditional scan.

Computation times are mainly dominated by the setup of the graph G_P; at worst, the whole optimization process takes a little more than 5 minutes, in the case of the b19 design which has 6000 FFs.

6. Conclusion and Perspectives

After giving some motivations for inserting scan at the RT-level, we have exposed what we believe is an important challenge for RTL-scan, which is finding a good stitching in one single pass, working only at the RT-level. Therefore, the purpose of the present work is to solve the following problem: given an RT-level description of the design (Verilog or Vhdl) we have to find a stitching ordering of the memory elements in the scan chain, which minimizes the impact of test circuitry on chip performance (area, power) and testing time. Then, we have motivated our choice of selecting wire length as the prime parameter to optimize.

To solve this problem, we proposed a new scan stitching algorithm for optimal scan chain insertion at RTL. The techniques used are derived from the combinatorial optimization and operations research domains. Indeed, our algorithm is divided into two main steps. The first step proposes a mathematical model describing the electronic problem. The second one offers a resolution methodology.

The model we propose is based on graph theory. To build it, we extract from the RTL description information on the proximity of the memory elements (existing paths between flip-flops, clock domains, and various other relations extracted from hierarchical analysis) and translate them in two graphs G_D and G_P using the Limited-BFS algorithm. Then, we have shown that the scan stitching decision problem is equivalent to the Traveling Salesman Problem and therefore *NP-Complete*. However, the 2-approximation for the TSP can be used to find near optimal solutions in polynomial time. From this, we developed our stitching algorithm.

Finally, we integrated our tool in an industrial design flow and performed experiments over several academic and industrial design benchmarks. Numerical evidence showed that we are able to limit such a cost due to scan insertion in a reasonable computing time, without impacting DFT quality, especially fault coverage. The method seems better than the traditional one for middle-sized designs. The industrial interest for our algorithms and tools is confirmed by our industrial partners. Our RT-level scan optimization algorithm has been incorporated into the tool HiDFT-SIGNOFF by DeFacTo Technologies. In case new flip-flops

Table 2: Characteristics of the graphs G_D and G_P.

Design	No. of FF	$\lvert V_D\rvert$	$\lvert E_D\rvert$	$\lvert V_P\rvert$	$\lvert E_P\rvert$
b09	28	401	719	28	378
b10	17	525	947	17	136
b11	31	1094	1845	31	465
b12	121	3879	6993	121	7260
b13	53	633	1042	53	1378
b14	245	27269	46778	245	29890
b15	449	33179	56798	449	100576
b17	1415	100358	171759	1415	697990
b18	3320	271320	462326	3320	2330318
b19	6642	531161	906201	6642	7061713
Simple-Spi	132	1427	8646	132	8778
Biquad	204	357	20706	204	20910
Ac-97	2289	18727	2381495	2289	707911

Table 3: Wirelengths and insertion times.

Design	No. of SC	Wirelength		Slack (%)	Insertion time (ms)	
		GL-S	RTL-S		P-graph	TSP
ITC 99 Benchmarks (VHDL)						
b09	2	2.06	1.63	−20.59	<1	<1
b10	2	1.94	2.06	5.97	10	<1
b11	3	4.77	5.03	5.38	20	10
b12	10	12.6	12.92	2.59	200	20
b13	5	3.39	3.32	−2.25	20	<1
b14	24	63.64	64.2	0.96	3 s	110
b15	24	126.3	122	−3.39	6 s	320
b17	70	395.6	370.4	−6.37	30 s	3 s
b18	300	1187.2	922.6	−22.29	3 m	10 s
b19	600	2329.6	1858.2	−20.24	5 m	20 s
Opencore designs (Verilog)						
Simple-Spi	10	11.5	10.4	−8.95	100	20
Biquad	20	34.1	31.3	−8.14	350	80
Ac-97	200	247.7	238.5	−3.71	30 s	8 s

Table 4: Fault coverage for RTL scan.

Design name	Fault coverage
ITC 99 Benchmarks (VHDL)	
b09	99.86%
b10	99.85%
b11	99.93%
b12	99.97%
b13	99.92%
b14	99.99%
b15	99.97%
b17	99.47%
b18	99.81%
b19	99.81%
Opencore designs (Verilog)	
Simple Spi	98.35%
Biquad	99.96%
Ac-97	99.80%

are added or removed to existing scan chains, it is important to goldenize the RTL code and reflect such changes directly into the RTL code. The DeFacTo tool HiDFT-SIGNOFF allows that by introducing the new scan architecture and by including the flip-flops new ordering. In this way, our work represents a progress in the state of the art, as previous works are not automated and/or were evaluated only for small designs.

Last but not least, an important contribution of our work is to make a neat separation between the DFT problem and the mathematical model. This separation allows the same software to work for several successive technology nodes.

Our initial results give rise to a number of new directions for further research. These are summarized below. First, one could investigate whether cumulating local and global optimization in the manner of [8] would improve our results. Another aspect that could be added to our model is the question of power during testing: can we take it into account without degrading ou results? Finally, design analysis at a

higher-level gives an opportunity to reduce both area overhead and test time application by adopting partial scan instead of full scan [16–18]. Deciding whether the combination of design analysis with our method could reduce even more the cost of scan would be worthwhile.

References

[1] B. Korte, L. Lovasz, H. J. Promel, and A. Schrijver, *Paths, Flows, and VLSI-Layout*, Springer, New York, NY, USA, 1990.

[2] S. H. Gerez, *Algorithms for VLSI Design Automation*, John Wiley & Sons, Chichester, UK, 1998.

[3] L. T. Wang, Y. W. Chang, and K. T. Cheng, *Electronic Design Automation: Synthesis, Verification, and Test*, Morgan Kaufmann, Boston, Mass, USA, 2009.

[4] Semiconductors Industry Association, *Test and Test Equipment. Update*, International Technical Roadmap for Semiconductor, 2010.

[5] L. T. Wang, C. E. Stroud, and N. A. Touba, *System-on-Chip Test Architectures: Nanometer Design For Testability*, Morgan Kaufmann, Boston, Mass, USA, 2008.

[6] S. Roy, "RTL based scan BIST," in *Proceedings of the VHDL International Users' Forum (VIUF '97)*, pp. 117–121, October 1997.

[7] C. Aktouf, H. Fleury, and C. Robach, "Inserting scan at the behavioral level," *IEEE Design and Test of Computers*, vol. 17, no. 3, pp. 34–42, 2000.

[8] Y. Huang, C. C. Tsai, N. Mukherjee, O. Samman, W. T. Cheng, and S. M. Reddy, "Synthesis of scan chains for netlist descriptions at RT-Level," *Journal of Electronic Testing*, vol. 18, no. 2, pp. 189–201, 2002.

[9] M. Hirech, J. Beausang, and X. Gu, "New approach to scan chain reordering using physical design information," in *Proceedings of the IEEE International Test Conference (ITC '98)*, pp. 348–355, October 1998.

[10] D. Berthelot, S. Chaudhuri, and H. Savoj, "An efficient linear time algorithm for scan chain optimization and repartitioning," in *Proceedings of the International Test Conference (ITC '02)*, pp. 781–787, Washington, DC, USA, October 2002.

[11] P. Gupta, A. B. Kahng, and S. Mantik, "Routing-aware scan chain ordering," *ACM Transactions on Design Automation of Electronic Systems*, vol. 10, no. 3, pp. 546–560, 2005.

[12] P. Gupta, A. B. Kahng, I. I. Măndoiu, and P. Sharma, "Layout-aware scan chain synthesis for improved path delay fault coverage," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 7, pp. 1104–1114, 2005.

[13] B. B. Paul, R. Mukhopadhyay, and I. S. Gupta, "Genetic algorithm based scan chain optimization and test power reduction using physical information," in *Proceedings of the IEEE Region 10 Conference (TENCON '06)*, Hong Kong, November 2006.

[14] K. D. Boese, A. B. Kahng, and R. S. Tsay, "Scan chain optimization: heuristic and optimal solutions," Internal Report CS Department, University of California at Los Angeles, Los Angeles, Calif , USA, 1994.

[15] S. Makar, "Layout-based approach for ordering scan chain flip-flops," in *Proceedings of the IEEE International Test Conference (ITC '98)*, pp. 341–347, October 1998.

[16] S. Bhattacharya and S. Dey, "H-SCAN: a high level alternative to full-scan testing with reduced area and test application overheads," in *Proceedings of the 14th IEEE VLSI Test Symposium*, pp. 74–80, Princeton, NJ, May 1996.

[17] T. Asaka, S. Bhattacharya, S. Dey, and M. Yoshida, "H-SCAN+: a practical low-overhead RTL design-for-testability technique for industrial designs," in *Proceedings of the IEEE International Test Conference (ITC '97)*, pp. 265–274, Washington, DC, USA, November 1997.

[18] R. B. Norwood and E. J. McCluskey, "High-level synthesis for orthogonal scan," in *Proceedings of the 15th VLSI Test Symposium*, pp. 370–375, May 1997.

[19] Private communication.

[20] L. Zaourar and Y. Kieffer, "Scan chain optimization for integrated circuits testing : an operations research perspective," in *Proceedings of the 23rd European Conference on Operational Research (EURO '09)*, p. 289, Bonn, Germany, July 2009.

[21] L. Zaourar, Y. Kieffer, C. Aktouf, and V. Julliard, "global optimization for scan chain insertion at the RT-level," in *Proceedings of the IEEE Computer Society Annual Symposium VLSI (ISVLSI '11)*, pp. 321–322, University Pierre and Marie Curie, Paris, France, July 2011.

[22] L. Zaourar, Y. Kieffer, and C. Aktouf, "An innovative methodology for scan chain insertion and analysis at RTL," in *Proceedings of the Asian Test Symposium (ATS '11)*, pp. 66–71, IEEE Computer Society, Washington, DC, USA, November 2011.

[23] W. J. Cook, W. H. Cunningham, and W. R. Pulleyblank, *Combinatorial Optimization*, Wiley-Interscience, New York, NY, USA, 1997.

[24] http://glaros.dtc.umn.edu/gkhome/views/metis/.

[25] http://www.cerc.utexas.edu/itc99-benchmarks/bench.html.

[26] http://www.opencores.org/.

Hindawi Publishing Corporation
VLSI Design
Volume 2012, Article ID 273276, 12 pages
doi:10.1155/2012/273276

Research Article

A Novel Framework for Applying Multiobjective GA and PSO Based Approaches for Simultaneous Area, Delay, and Power Optimization in High Level Synthesis of Datapaths

D. S. Harish Ram,[1] M. C. Bhuvaneswari,[2] and Shanthi S. Prabhu[1]

[1] *Department of Electronics and Communication Engineering, Amrita Vishwa Vidyapeetham University, Coimbatore 641112, India*
[2] *Department of Electrical and Electronics Engineering, PSG College of Technology, Coimbatore 641004, India*

Correspondence should be addressed to D. S. Harish Ram, ds_harishram@cb.amrita.edu

Received 3 February 2012; Revised 19 August 2012; Accepted 23 September 2012

Academic Editor: Dinesh Mehta

Copyright © 2012 D. S. Harish Ram et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

High-Level Synthesis deals with the translation of algorithmic descriptions into an RTL implementation. It is highly multi-objective in nature, necessitating trade-offs between mutually conflicting objectives such as area, power and delay. Thus design space exploration is integral to the High Level Synthesis process for early assessment of the impact of these trade-offs. We propose a methodology for multi-objective optimization of Area, Power and Delay during High Level Synthesis of data paths from Data Flow Graphs (DFGs). The technique performs scheduling and allocation of functional units and registers concurrently. A novel metric based technique is incorporated into the algorithm to estimate the likelihood of a schedule to yield low-power solutions. A true multi-objective evolutionary technique, "Nondominated Sorting Genetic Algorithm II" (NSGA II) is used in this work. Results on standard DFG benchmarks indicate that the NSGA II based approach is much faster than a weighted sum GA approach. It also yields superior solutions in terms of diversity and closeness to the true Pareto front. In addition a framework for applying another evolutionary technique: Weighted Sum Particle Swarm Optimization (WSPSO) is also reported. It is observed that compared to WSGA, WSPSO shows considerable improvement in execution time with comparable solution quality.

1. Introduction

Data path algorithms can be depicted using Data Flow Graphs (DFGs). Each *node* in the DFG represents an operation that is to be executed in the algorithm (Figure 1). The operation may be arithmetic such as addition or multiplication or logical. High Level Synthesis (HLS), also known as Behavioral Synthesis is the process of converting an algorithmic description into a synthesizable Register Transfer Level (RTL) netlist. The high level description may be in the form of a programming language such as C or DFGs. The HLS design flow consists of three subtasks, namely, scheduling, allocation, and binding. *Scheduling* determines the *time step* at which a node in the DFG is executed. For example, in the following DFG, the node *d* is scheduled in the second time step. Some nodes have *mobility* in that they can have several potential execution instances as in the case of node c which can execute in time step 1 or 2 without affecting the schedule.

The term allocation refers to the process of assigning resources or Functional Units (FUs) to execute a particular operation. For instance, one possible allocation for the DFG shown in Figure 1 is three adders and one multiplier. In *binding*, a node is bound to a FU for execution. Thus, node *a* may be bound to Adder 1, node *b* to Adder 2 and node *c* to Adder 3. The HLS sub-tasks can be performed in any order or simultaneously. The result of each sub-task influences the others.

The main purpose of HLS algorithms is to facilitate an early design space exploration of various alternative RTL implementations for a particular algorithm with the primary objective of minimizing area, delay, and power of the final design. This paper investigates the use of evolutionary techniques such as Genetic Algorithms (GAs) and Particle

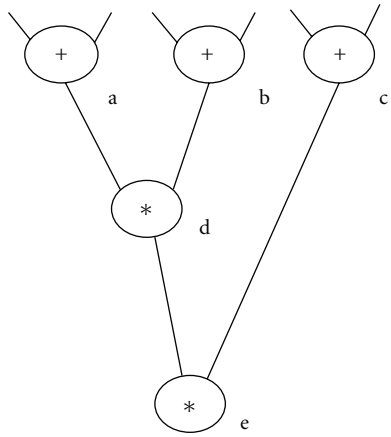


Figure 1: A Data Flow Graph.

Swarm Optimization (PSO) for optimal scheduling of DFGs during High Level Synthesis. Since the problem is multi-objective in nature, the GA framework is adapted for applying true multi-objective algorithms such as NSGA II that yield a population of Pareto optimal solutions with each solution representing a trade-off among the objectives. Also we have adapted the framework developed for applying another multi-objective evolutionary approach, Weighted Sum PSO (WSPSO), for solving the same problem.

This work is based on the technique reported by the authors in Harish Ram et al. [1]. The methodology described in [1] proposes an NSGA II based approach for optimal scheduling during HLS of datapaths. The technique has been validated on additional benchmarks. Also the basic framework used for implementing the GA has been customized for applying WSPSO to the multi-objective HLS problem.

Previous work on multi-objective optimization in HLS has focused primarily on area and delay minimization. In [2], a weighted sum approach has been proposed for area and delay minimization during HLS of DFGs. In Ferrandi et al. [3], the authors have used two different encodings which prioritize scheduling and binding, respectively. The cost function is computed using regression models derived from actual characterizations. A true multi-objective GA (NSGA II) is used to optimize the schedules. We propose an evolutionary framework using the chromosome encoding in [2], with the cost function modified to incorporate power in addition to area and delay. Power cost of a schedule is determined as the likelihood of the schedule to yield low-power bindings, computed from compatibility graph metrics described in [4]. The methodology is evaluated for WSGA and NSGA II on standard benchmarks [2, 5] and the superiority of NSGA II is demonstrated. The framework is extended for applying WSPSO to the HLS scheduling problems.

The rest of the paper is organized as follows. Section 2 describes the new algorithms used in the work. Section 3 gives a review of related literature. Section 4 outlines the evolutionary encoding scheme and cost function computation used in the work. Sections 5, 6, and 7, respectively detail the WSGA, NSGA II, and WSPSO based methodologies. Section 8 discusses the results and Section 9 concludes the paper.

2. New Algorithmic Techniques Used

The application of Genetic Algorithms and similar meta-heuristics in engineering problems has come to stay and is a well researched domain. However, many optimization problems are multi-objective in nature with the candidate solutions trading off one objective in favor of the other [6]. A classic instance in the VLSI domain is the area-delay trade-off. In such problems, the focus is on identifying a population of Pareto optimal solutions rather than a single optimal solution. A simple approach in solving such problems will use a weighted cost function involving all the objectives to be optimized. However this technique is flawed since it yields solutions with limited diversity [6]. This necessitates the development of true multi-objective algorithms that are capable of exploring the entire design space of solutions with different trade-offs among the objectives. Several multi-objective techniques have been proposed to solve engineering problems in the past decade and a half [7–9]. Multi-objective optimization of area and delay during datapath synthesis has been addressed in [2, 3]. However power is not considered in these schemes. In [10] Bright and Arslan had proposed a GA based approach to generate Pareto surfaces for area and delay during synthesis of DSP designs. The paper does not describe any scheme for ensuring the diversity of the solutions. We propose a multi-objective methodology using NSGA II [7] for power, area, and delay optimization of DFGs. NSGA II incorporates a selection technique based on the crowding distance of individuals in a population that preserves the diversity of the solutions when the algorithm evolves. The technique is evaluated on standard benchmarks and fares substantially better than WSGA in terms of solution quality as well runtime. PSO is another recent evolutionary technique that has been widely investigated for solving intractable problems in engineering and science [8, 11]. PSO has been widely applied in scheduling problems such as instruction scheduling and traveling salesman problem (TSP) [12–16]. We have extended the chromosome encoding scheme and cost function computation used in our GA technique for applying Weighted Sum PSO for datapath synthesis.

3. Review of Previous Work

Early work in Behavioral Synthesis focused on *constructive approaches* as in force directed scheduling [2, 17]. These are essentially greedy approaches vulnerable to local minima. *Transformational approaches* [2] start with an initial schedule and apply transformations to improve the initial solution. However, the quality of the solutions largely depends on the transformations used and the heuristics used to select between applicable transformations [2]. *Graphical approaches* have been proposed in which the minimum

cost binding problem is converted into a multiway partitioning of a flow graph. In [18], a graph network is used to model the binding problem with the edge weights representing the switching cost of executing two nodes in succession on the same functional unit (FU) or register. Thus the problem reduces to a multi-way partitioning of the graph for minimal flow cost. This method is reported for optimal bus scheduling and can be extended to FUs also. Scheduling under constraints is not addressed here. In [19], graph based heuristics are proposed to choose an optimal set of resources for a scheduled DFG with more than one architecture to choose from for each functional unit. *Mathematical approaches* formulate the synthesis problem using Integer Linear Programming (ILP) approach or some other mathematical optimization tools such as game theory. Simultaneous binding and scheduling of a DFG using ILP is addressed in [20]. The precedence and time constraints are built into the ILP formulation. The cost function is evaluated based on signal statistics of the inputs obtained through a one-time simulation. The technique suffers from the drawback that ILP methods do not scale well for large circuits and may have to be combined with some heuristics. An ILP approach is combined with retiming for power optimization in [21]. In [22], a game-theoretic approach for power optimization of a scheduled DFG is proposed. The functional units are modeled as bidders for the operations in the DFG with power consumption as the cost. The algorithm does not scale well for larger number of functional units since the complexity increases exponentially with the number of players (bidders).

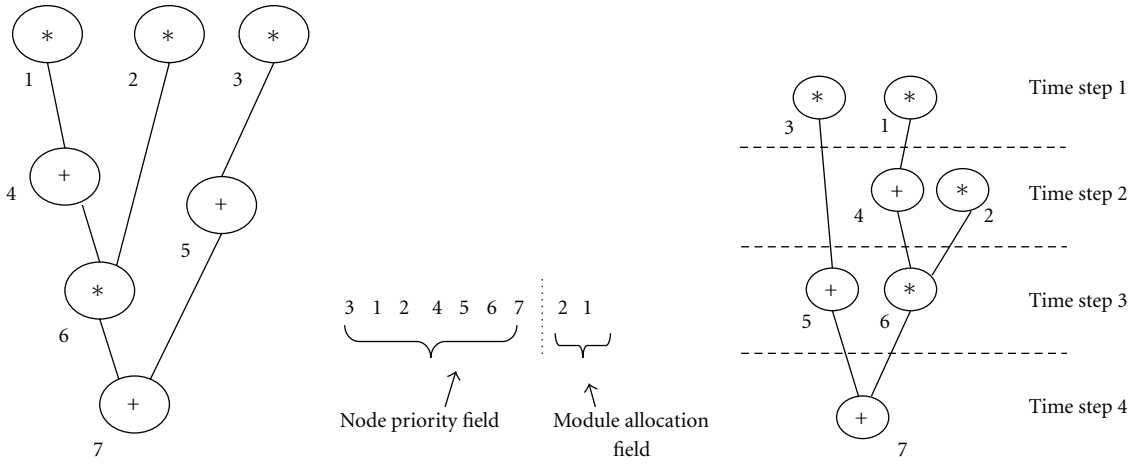
The use of Genetic Algorithms (GAs) for optimal scheduling, allocation, and binding in Behavioral Synthesis has been widely reported in the literature. A survey of GA based methodologies for Behavioral Synthesis is presented in [2]. Genetic Algorithms being population based heuristics are extremely effective in design space exploration. A GA based methodology for datapath synthesis from a DFG for multi-objective area and delay optimization is described in [2]. The authors propose a multichromosome encoding scheme for the DFG scheduling priority and functional unit constraints. List scheduling is carried out based on the priorities coded in the chromosome. A weighted cost function incorporating both area and delay is employed for assessing the fitness of a given schedule. But the weighted sum approach suffers from the drawback that in a sufficiently nonlinear problem; it is likely that the optimal solutions resulting from a uniformly spaced set of weight vectors may not result in a uniformly spaced set of Pareto optimal solutions [6]. Ferrandi et al. [3] have proposed an approach based on Multi-objective GA using the algorithm NSGA II [6, 7]. The authors have used two different encoding schemes. The priority based scheme [2] arranges nodes in the DFG in the order in which they have to be scheduled by a list scheduler whereas the binding based scheme [23] incorporates binding information pertaining to each DFG node. The area and delay costs are extracted from a regression model built using actual characterizations. Power is not included in the fitness evaluation.

The optimal scheduling problem in HLS is somewhat analogous to the traveling salesman problem (TSP). In both cases, an evolutionary approach necessitates a chromosome encoding which specifies the *order* in which scheduling of nodes (cities to be visited in case of TSP) is carried out as described in the next section. These problems differ in their cost functions. TSP seeks to minimize the distance travelled whereas in HLS the objective is to optimize area, delay, or power as the case may be. In addition, the precedence of execution of each node is to be considered during HLS. Wang et al. propose a PSO methodology for TSP in [12]. The authors describe a technique for generating new scheduling strings using the swap operator. An encoding scheme for efficient convergence of the particles during PSO is proposed in [13]. In [14], Jie et al. describe a scheme for reducing the scale of the problem for large number of nodes, without affecting exploration of the search space. To our knowledge, PSO has not been used in multi-objective High Level Datapath Synthesis.

4. Evolutionary Encoding Scheme, Cost Function Computation, Crossover, and Mutation

4.1. Encoding Scheme. A multi-chromosome encoding scheme reported in [2] is used in the proposed technique. The same encoding is used for both GA and PSO. In this scheme a chromosome has a *node scheduling priority field* and a *module allocation field* (Figure 2(b)). The structure of the chromosome is such that simultaneous scheduling of a DFG and functional unit allocation can be carried out. The DFG nodes are scheduled using a list scheduling heuristic [2, 17]. The nodes are taken up for scheduling in the order in which they appear in the chromosome. The module constraint is described in the *module allocation field*. The respective constraints of the number of multipliers and adders are specified in this field. If additional FUs such as subtractors are present, the module allocation field will have more terms.

As an example, an unscheduled DFG and a corresponding chromosome encoding are shown respectively in Figures 2(a) and 2(b). The scheduling starts with the first time step. Node 3 which appears first in the chromosome is scheduled in the first time step itself. Also node 1 is scheduled to be executed in time step 1 since it appears next in the chromosome. Node 2 which is next cannot be scheduled in the first time step because only two multipliers are available as specified in the module allocation field. Therefore Node 2 has to wait till the next time step. Node 4 which performs addition cannot be scheduled in this time step since its predecessor node, that is, Node 1, has just been scheduled. The result of the execution of this node will be available only in the next time step. Similarly nodes 5, 6, and 7 cannot be scheduled since the results of their predecessor nodes are not ready. Thus we move to the second time step since no further scheduling is possible in the current time step. In this step, nodes 2 and 4 which were not scheduled in the previous time step are scheduled for execution. This sequence continues till



(a) Unscheduled DFG (b) Multi-chromosome (c) Scheduled DFG

Figure 2: Multichromosome and corresponding schedule.

all nodes are scheduled. It has to be noted that in a valid string the precedence relationship between the nodes has to be maintained. For instance, in the chromosome shown in Figure 2(b), the nodes 1 and 2 which are the predecessors for node 6 appear before node 6 in the priority list. The scheduled DFG is shown in Figure 2(c).

4.2. Cost Function Computation. The fitness evaluation of each chromosome is carried out as described in [2] for area and delay. The potential of a given schedule to yield low-power bindings is determined from the compatibility graphs for the FUs and registers extracted from the schedule using a method described in [4]. Actual power numbers are not computed.

4.2.1. Area and Delay Cost. The delay or length L is the number of time steps or clock cycles needed to execute the schedule. For example $L = 4$ for the schedule in Figure 2(c). The area cost A of the schedule is based on the total FUs and registers required to bind all the nodes in the DFG. The former is known from the chromosome itself (e.g., 2 multipliers and 1 adder for the DFG in Figure 2(c)). The latter is obtained by determining the total number of registers required for the DFG implementation using the left edge algorithm [17]. The total area is expressed as the total number of transistors required to synthesize the FUs and registers to a generic library.

4.2.2. Power Cost. The potential of an individual schedule to yield a low-power binding solution is found out using a set of metrics described in [4]. Here a *compatibility graph* (CG) is extracted from the scheduled DFG corresponding to the schedule obtained from the node priority field in each chromosome of the population. The compatibility graph will have an edge between two nodes if they are *compatible.* Two nodes are said to be compatible if they can be bound to the same FU or register. For instance, nodes 1 and 2 in the DFG in Figure 2(c) are FU compatible since they execute the same

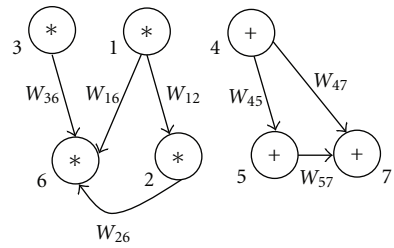


Figure 3: Compatibility graph.

operation (multiplication) and their lifetimes do not overlap. On the other hand, nodes 3 and 2 though executing the same operation are not compatible since their lifetimes overlap. The output of node 3 is required by node 5 in the third time step whereas node 2 has to be executed in the second time-step itself. The compatibility graph for the schedule given in Figure 2(c), is given in Figure 3. Separate CGs are created for the registers and FUs. The edge weights between two nodes represent the *switching cost* when these two nodes are executed one after the other in the same FU [24].

The power-related metrics [4] are based on the edge weights of the compatibility graph and are defined as follows:

m_1 = total number of edges in the graph,

m_2 = average of edge weights for the lowest k% in the value range for each node where k is user-defined

m_3 = average of all edge weights (i.e., m_2 for k = 100%). A DFG may yield a low-power binding if m_1 is high and m_2 and m_3 are low. The power cost function designed based on the compatibility graph metrics is given below

$$P = \left(\frac{n}{m_1}\right) + m_2 + m_3, \tag{1}$$

where n is a tuning parameter, is chosen depending on the value of m_1, and has a value greater than m_1. The rationale

for the choice of the power metrics is as follows. Higher number of edges (m_1) in the CG indicates higher number of possible bindings and hence more likelihood of the schedule yielding low-power bindings. Thus the power metric is made *inversely proportional* to m_1. Smaller values of CG edge weights indicate lesser switching cost and hence lower power dissipation when a pair of nodes execute on an FU. Thus a schedule having a CG with smaller average edge weight has better potential to yield more power-aware bindings. This dependency is encoded in the second and third terms of the cost function, m_2 and m_3.

4.3. Crossover. We have used the single-point crossover technique reported in [2]. In this method, the precedence relationships between the nodes in the node priority field of the multi-chromosome are maintained. The technique is illustrated in Figure 9.

The parent string is retained intact till the crossover point. For instance, in the example shown in Figure 9, crossover is carried out from the fifth position in the string. Thus, in offspring 1, the substring 3 1 2 4 is replicated from Parent 1. From location 5 onwards, the Parent 2 string is traversed and those nodes in the list that are *not present* are filled in the same order in which they appear in Parent 2. For instance, the nodes 1, 2, and 4 are already present in the substring 3 1 2 4 taken from Parent 1, but 6 is not present and hence is filled in the location 5. The node 3 which appears next is already present whereas 5, which appears next to 3, is not present and is filled in. The procedure is continued till the entire string is generated. The same sequence is repeated with Parent 2 for generating offspring 2.

For the module allocation field, the module allocations which appear after the crossover point are swapped. In Figure 9, the module allocation strings are 2 1 and 2 2 respectively for parent *Parent 1* and *Parent 2*. After crossover, the module allocation strings become 2 2 and 2 1, respectively.

4.4. Mutation. Mutation also is based on the technique described in [2]. For the node priority field, a node in the list is chosen at random and is moved without affecting the node precedence constraints. As an example, consider the same chromosome described previously, that is, the string 3 1 2 4 6 5 7 | 2 1. Let node 5 in location 6 be chosen for the mutation. Its predecessor is node 3 in the first slot and its successor is node 7 in the seventh slot. Thus the node can be moved to any slot from slot 2 to slot 6 without violating the precedence constraints. The actual slot to which the node is to be moved is chosen by generating a random number between 2 and 6. If the random number generated is, say, 3, then the node 5 is moved to slot 3 and the new chromosome after mutation becomes 3 1 **5** 2 4 6 7 | 2 1 (mutated node is shown in bold).

For the module allocation field, the mutation is effected by simply incrementing or decrementing the number of functional units. For example, we may decrement the number of FU in the above multi-chromosome. Thus after mutation, the final chromosome string becomes 3 1 **5** 2 4 6 7 | **1 1** (mutated locations are shown in bold).

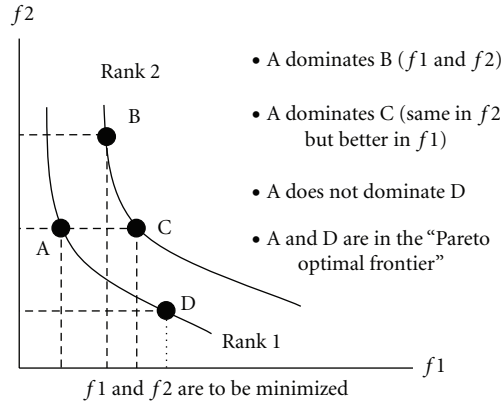


Figure 4: Multi-objective solution space and Pareto Front.

5. Weighted Sum GA

The weighted sum approach [6] used in this paper converts a multi-objective problem into a single objective GA. Each term in the cost function of this GA represents the cost of one objective to be optimized and is multiplied by a weight factor between 0 and 1. This weight is chosen depending upon the extent to which a particular objective is to be optimized. This approach is computationally simple. The fitness evaluation of each individual is based on a cost function value calculated using a weighted sum approach. The cost function described in [2] is modified to include a power dissipation term described in the previous section and is given below:

$$C = \frac{w_1 L_s}{L_{\max}} + \frac{w_2 A}{A_{\max}} + w_3 P, \tag{2}$$

where w_1, w_2, and w_3 are the weights of the area, delay, and power terms, respectively, with w_1, w_2, and w_3 always obeying the relation $w_1 + w_2 + w_3 = 1$. The terms L_s and A are the length and area, respectively, of the implementation of the given schedule and P is the power metric. $A_{\max}$ and $L_{\max}$ are the maximum area and delay, respectively, among solutions in the current population. A population size of 100 is used in the GA implementation. Elitism is used for preserving the best solutions in a generation. Binary tournament selection is used to identify parents for crossover. The crossover probability used is 0.9 and mutation probability is 0.2. The WSGA run was repeated with 10 different initial seed populations.

6. NSGA II

In a *true multi-objective* algorithm, the solutions converge to the *true* Pareto front of *non-dominated* solutions. A solution is said to be *nondominated* if no other solution can be found in the population that is better for *all the objectives* (Figure 4). A multi-objective solution space for a minimization problem involving two objectives and the Pareto front is depicted in Figure 4. The weighted sum approach described in the previous section suffers from lack of diversity in that uniformly spaced weights do not result in a uniform spread of solutions as shown in Figure 5. The

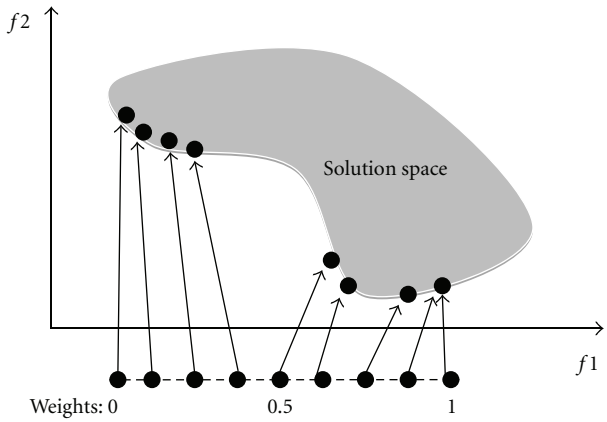


Figure 5: Solutions for WSGA [6].

figure depicts a solution space for a minimization problem with two objectives, $f1$ and $f2$. The x-axis represents different values of $f1$ for uniformly spaced values for the weight for $f2$ (say, w_2). For $w_2 = 0$, $f1$ has the most optimal value and $f2$ the worst. For higher values of w_2, the objective $f2$ shows progressively better values by trading off $f1$. But it can be observed that the values of $f2$ obtained by trading off $f1$ are not uniformly spaced though the weight for $f2$ (w_2) is increased in equal steps. Besides, proper assignment of weights and, hence, solution quality depend on knowledge of the solution space [6].

Deb proposed the "Non-dominated Sorting GA-II" or NSGA II [7], which is a true multi-objective GA. It uses the notion of crowding distance to ensure diversity among the solutions in a population. Initially a random seed is created. The cost of each objective for all the solutions is determined and they are classified into ranks based on nondomination. The Rank I individuals are fully non-dominated whereas those in Rank 2 are dominated by the Rank I individuals and so on. Each solution is assigned a fitness based on its rank and crowding distance. The crowding distance is a measure of the uniqueness of a solution. Crossover and mutation are performed on the individuals using the method described in [2]. The parents and offspring in a particular generation are merged and the individuals for the next generation are selected based on the crowding distance metric [6, 7]. Selection of individuals with higher crowding distance is favored for better diversity among solutions. The population size in a generation was 100. The GA was run for 200 generations. A flow diagram depicting the NSGA II methodology is shown in Figure 6. A detailed discussion of the results obtained for this method on standard DFG benchmarks is given in Section 8.

7. Weighted Sum Particle Swarm Optimization (WSPSO)

7.1. Introduction. Particle Swarm Optimization (PSO) is an evolutionary approach proposed by Eberhart and Kennedy, inspired by the behavior of bird flocks or schools of fish. PSO, like GA, operates on a population of candidate solutions referred to as a *swarm*. Each solution in a swarm is called a *particle*. The swarm is made to evolve by applying a *velocity* to each particle. The best solution in each swarm is called the "particle best" or *pbest* and the best solution among all the swarms so far is the "global best" or *gbest*. When the swarm moves, each particle is subjected to a velocity which tends to propel it in the direction of *pbest* as well *gbest* with each direction being assigned a weight as modeled by the equation given below:

$$v^i_{j,t+1} = wv^i_{j,t} + \eta_1 R_1\left(p^i_{j,t} - x^i_{j,t}\right) + \eta_2 R_2\left(g^i_{j,t} - x^i_{j,t}\right), \quad (3)$$

where $v^i_{j,t+1}$ = velocity of the ith particle for the $t+1$th iteration, and $v^i_{j,t}$ = velocity of the ith particle for the tth iteration.

The weight w assigns some *inertia* to the particle. The parameters η_1 and η_2 assign weights to the *pbest* and *gbest* variables, that is, the extent to which these positions influence the movement of the particle. The random values R_1 and R_2 introduce a degree of perturbation in the particle, for better exploration of the search space.

7.2. Weighted Sum PSO (WSPSO) for DFG Scheduling. Scheduling of DFGs during datapath synthesis is a good candidate for application of PSO. This problem is somewhat analogous to TSP [12, 16], albeit with precedence and module constraints. The multi-chromosome encoding introduced in Section 4 can be used for PSO also. In discrete problems, the notion of velocity is implemented by adapting the crossover technique [2, 16] for the PSO problem. A particle in the swarm is represented by a single multi-chromosome string representing a given DFG schedule and allocation. This particle is crossed over with the current particle that represents the *gbest* (or *pbest*) to implement the velocity function. The particle is crossed over with both the *gbest* and *pbest* using the procedure outlined in Section 4. This is illustrated in Figure 10.

The same approach is followed for *pbest* also. It can be seen that the degree of shift of the particle towards the *pbest* or *gbest* can be controlled by appropriately choosing the crossover location. If the location is in the beginning of the string then the shift is more.

Area, delay, and power cost of the schedule represented by the chromosome is computed using the method described in Section 5. The weighted sum approach outlined in Section 6 is used to determine the fitness of a particle represented by the multi-chromosome. A swarm size of 100 individuals was used. The total number of generations (swarms) was 70. The flowchart shown in Figure 7 summarizes the WSPSO based methodology used.

The PSO methodology developed was tested on various benchmarks. It was found to exhibit better runtimes than WSGA with comparable solution quality.

8. Results and Discussion

All the results reported below were obtained from executing the algorithms in an Intel i5-2400 CPU with a clock frequency of 3.10 GHz and 4 GB RAM.

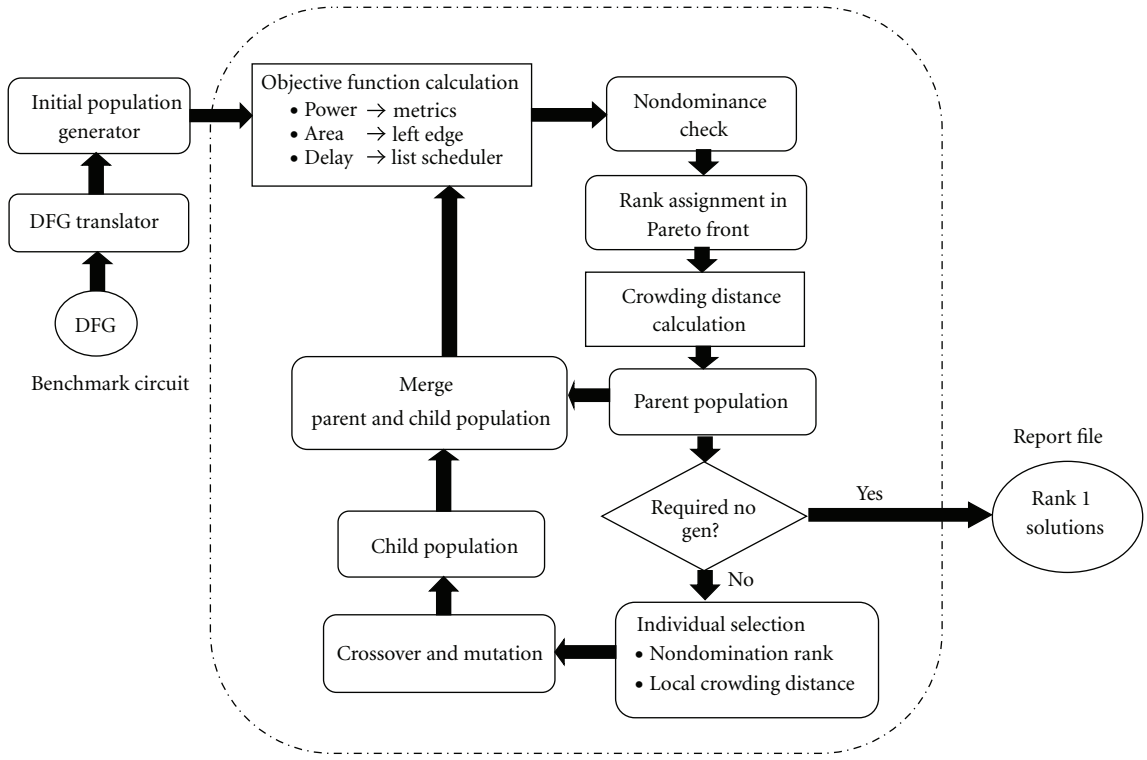


Figure 6: NSGA II based methodology.

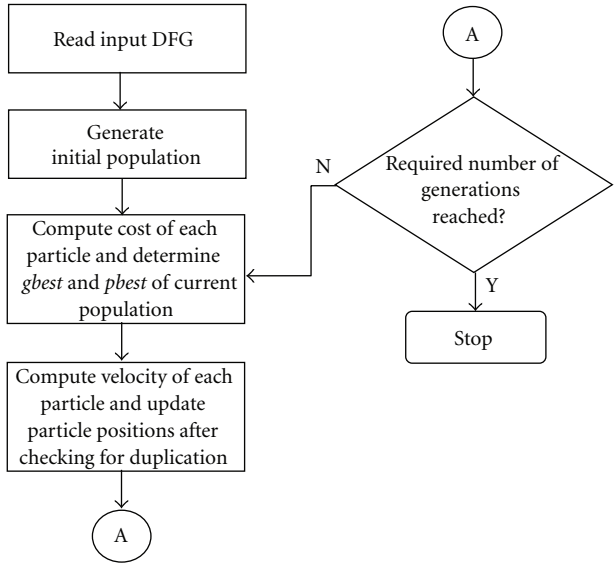
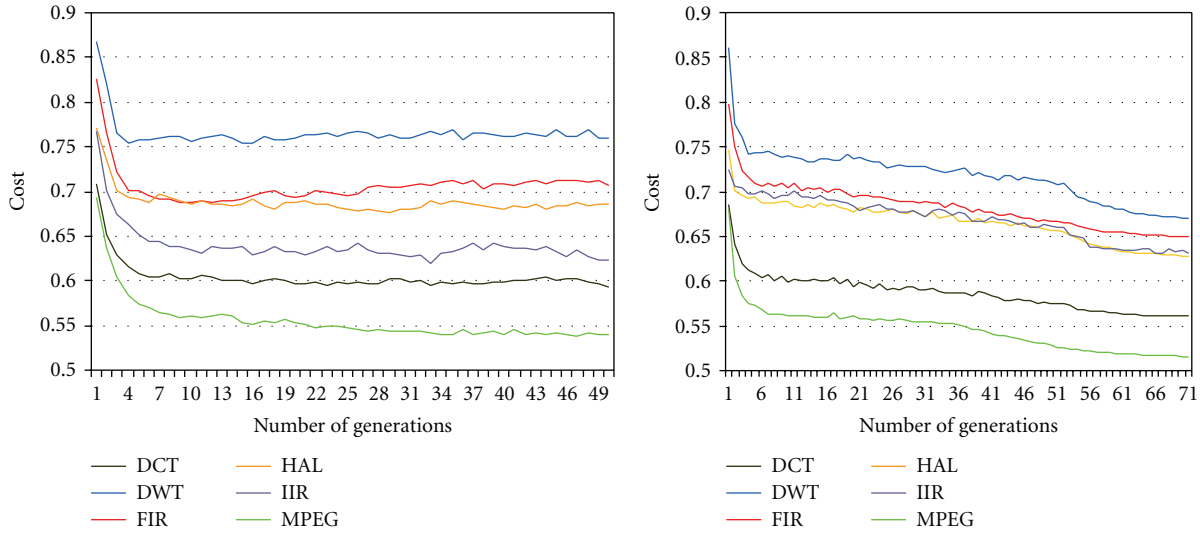


Figure 7: PSO methodology for DFG scheduling.

8.1. WSGA and WSPSO Convergence. The convergence of WSGA and WSPSO is verified by plotting the cost function with respect to the number of generations for each DFG benchmark (Figures 8(a) and 8(b)). For WSGA, the number of runs was 50 and for WSPSO the algorithms were run for 70 generations. The convergence plots for each benchmark are presented in Figures 8(a) and 8(b) respectively for WSGA and WSPSO. It is observed from the cost function values at convergence that the extent of optimization observed for all the benchmarks is comparable for both WSGA and WSPSO. This is further discussed in connection with the results in Table 6.

8.2. Comparison of Power Aware WSGA with GA without Power in the Cost Function [2]. One of the contributions of this work is the introduction of a power metric that indicates at an early stage the *likelihood* of a schedule to yield a low power binding solution during the binding phase. This metric is added to the weighted sum cost function for area and delay optimization proposed in [2]. The efficacy of the modified GA with power is verified by evaluating the modified WSGA on various benchmarks and comparing with the results of WSGA method reported in [2] which does not incorporate power in the cost function. The weight assigned to the power cost is 0.7 and the other objectives are assigned equal weights of 0.15 each.

It was observed that the modified GA methodology yielded schedules that had improved values of the power metric indicating higher likelihood of obtaining low power bindings. The results of the comparison are listed in Table 1. An average reduction of around 9% is observed in the power metric for the power aware GA run. The reduction is not significant for the IIR benchmark which has only 9 nodes. Hence the search space of feasible schedules is limited. The HAL benchmark though having only 10 nodes yields a better power number since it has higher mobility for the nodes and hence more number of feasible schedules. The power



(a) WSGA convergence

(b) WSPSO convergence

Figure 8: Convergence plot for WSGA and WSPSO.

Table 1: Comparison of WSGA and power-aware WSGA.

Benchmark	WSGA without power optimization [2]			Power-aware WSGA			Reduction in power cost metric
	Power metric	Area (register units)	Delay (time steps)	Power metric	Area (register units)	Delay (time steps)	
IIR	0.9062	42.9	5.44	0.8996	48.48	5.15	0.73%
HAL	0.9911	38.33	4.74	0.8719	37.81	4.98	12%
FIR	0.6487	42.73	11.63	0.6029	54.48	11.32	7%
MPEG	0.6317	81.5	8.50	0.6046	107.5	9.00	17%

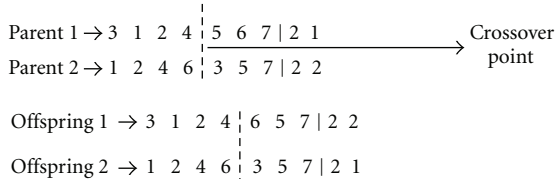


Figure 9

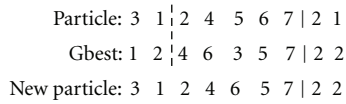


Figure 10

aware WSGA run yields better power optimal solutions for the FIR and MPEG benchmarks which have 23 and 28 nodes, respectively, obviously due to the higher number of nodes and also more degrees of freedom in moving the nodes for generating different schedules.

Table 2: Metrics evaluating closeness to the Pareto optimal front for IIR benchmark for NSGA II.

Performance metric	WSGA	NSGA II
Error ratio	0.7143	0
Generational distance	0.02708	0
Maximum Pareto optimal front error (MFE)	0.1833	0
Spacing	0.1046	0
Spread	1.0026	0
Weighted metric	0.5148	0

Table 3: Metrics evaluating closeness to the Pareto optimal front for HAL benchmark for NSGA II.

Performance metric	WSGA	NSGA II
Error ratio	0.7826	0
Generational distance	0.0050	0
Maximum Pareto optimal front error	0.1832	0
Spacing	0.0516	0
Spread	1.8600	0
Weighted metric	0.9360	0

8.3. NSGA II Results and Analysis. The quality of solutions obtained using a multi-objective algorithm must be assessed

Table 4: Comparison of WSGA and NSGA II on standard benchmarks.

Benchmark	WSGA			NSGA II		
	Power metric	Area (register units)	Delay (time steps)	Power metric	Area (register units)	Delay (time steps)
IIR	0.9019	48.48	5.148	0.9063	35.41	5.88
HAL	0.9901	37.81	4.981	0.9923	39.91	4.75
DWT	0.7126	50.92	10	0.6821	48.95	10
FIR	0.6416	54.48	11.32	0.6312	43.58	11.23
MPEG motion vector	0.6714	107.5	9	0.6561	63.13	10.71
DCT	0.6086	79.06	12.18	0.6087	77.44	12.12

Table 5: Execution times in seconds.

Benchmark circuit	WSGA	NSGA	% reduction
IIR	6	4	33.33%
HAL	11	11	0%
DWT	26	5	80.77%
FIR	112	9	91.96%
MPEG motion vector	48	23	52.08%

in terms of the spread of the solutions and the closeness of the individuals to the true Pareto front. The metrics described in [6] for diversity, spread and Pareto front errors were computed for the Rank I individuals obtained for the IIR filter and HAL benchmarks [2, 5] using WSGA and NSGA II. For computing the quality metrics, the true Pareto front was obtained for these two benchmarks by exhaustive search. The metric values obtained for the WSGA and NSGA runs are listed in Tables 2 and 3. The *error ratio* is the fraction of solutions in the population of Rank I individuals, which are not members of the true Pareto-optimal set, P^*. The *generational distance* metric gives a measure of the average distance of the solutions from P^*. The *maximum Pareto front error* gives the distance of the solution which is farthest from the Pareto front. The values of the metrics, *spacing* and *spread* indicate the diversity of solutions obtained. The *weighted metric* is the weighted sum of the closeness and diversity measures. These metrics should have small values for a good multi-objective algorithm. The results of comparing NSGA II and WSGA in terms of the quality metrics are given in Tables 2 and 3, respectively, for the IIR and HAL benchmarks.

The above metrics indicate the superior quality of the NSGA II solutions over those obtained by the Weighted Sum GA approach. All metrics have zero values for NSGA indicating that the solutions are *fully coincident* with the true Pareto front individuals.

The NSGA II methodology was evaluated on several standard DFG benchmarks and the results are summarized in Tables 4 and 5. Table 4 shows the average values of the delay, area, and power metrics obtained for the WSGA and NSGA II runs. Power dissipation is quantified in terms of the potential of the schedule to yield low power bindings as described in Section 4. Area is computed as the total gate count of the FUs and registers when synthesized to a generic library and is normalized to the register gate count as register units. Delay is expressed in terms of the total time steps required for completing each schedule. The average values for the NSGA II run are either better than WSGA or comparable in most of the cases. For the smaller DFGs (IIR and HAL), the NSGA II results do not exhibit an appreciable improvement in the *average* values. This is because of the fewer number of feasible solutions. Since NSGA searches the solution space more efficiently the effect of trade-offs between the three objectives is more pronounced. Another aberration that is noticeable is in the case of the MPEG motion vector benchmark where a substantial reduction in average area is noticed at the expense of higher average delay. This DFG is rich in multiplication nodes whose implementation is area intensive. Hence an effective search of the solution space will yield solutions that exhibit tremendous trade-offs between delay and area. Thus a considerable reduction in average area is observed at the expense of delay. Thus it can be concluded that NSGA II, being a true Multi-Objective GA, is highly effective in more intensive exploration of the design space. This is evident from the quality metrics as well as the results on the various DFG benchmarks.

Table 5 lists out the comparison of the execution times. The NSGA II algorithm shows appreciable improvement over WSGA with an average reduction of 51.63% in execution time. As expected, the improvement is more noticeable in the case of the larger benchmarks which necessitate exploration of a much larger search space. The search space is multidimensional and nonlinear and hence increase in number of DFG nodes will add exponentially large complexity to the search process.

8.4. WSPSO—Results and Analysis. The WSPSO-based weighted sum approach outlined in Section 7 was evaluated on the DFG benchmarks and compared with WSGA. A particle size of 100 was used and the WSPSO was run for 70 generations. The average power, area, and delay values of 10 runs of the algorithm with different initial seed populations are tabulated for various DFG benchmarks. The quality of the solutions was also assessed against the results from exhaustive search. The results are shown in Tables 6 and 7. Initial results are comparable to the corresponding results for WSGA. However, the quality and run times can be improved upon optimizing the swarm size and introducing additional operators in the velocity function used. Further investigations need to be carried out in this direction.

Tables 6 and 7 present the comparison of WSGA and WSPSO on standard DFG benchmarks. Whereas Table 6

Table 6: Comparison of WSGA and WSPSO on DFG benchmarks.

Benchmark	WSGA				WSPSO				Reduction in execution times%
	Power	Area (register units)	Delay (time steps)	Execution time (seconds)	Power	Area (register units)	Delay (time steps)	Execution time (seconds)	
IIR	0.9019	48.48	5.148	6	0.9083	35.93	5.86	4	33.33%
HAL	0.9901	37.81	4.981	11	0.9960	33.75	4.98	8	27.27%
DWT	0.7126	50.92	10	26	0.7242	26.57	10.11	14	46.15%
FIR	0.6416	54.48	11.32	112	0.6595	44.68	10.59	97	13.39%
MPEG motion vector	0.6714	107.5	9	48	0.6984	95.30	7.69	34	29.17%
DCT	0.6086	79.06	12.18	1528	0.6120	75.98	11.30	1408	7.85%

Table 7: Metrics evaluating closeness to the Pareto optimal front for IIR benchmark for WSGA and WSPSO.

Performance metrics	WSGA	WSPSO
Error ratio	1	1
Generational distance	2.09	3.02
Maximum pareto optimal front error (MFE)	81	43.02
Spacing	2.73	0
Spread	1.56	1
Weighted metric	1.82	2.04

compares average values of power, area, and delay and execution times, Table 7 lists out the performance metrics [6] that indicate the quality of the solutions obtained in terms of diversity and closeness to the true Pareto front. This is assessed by generating the true Pareto front using exhaustive search and comparing the Rank I solutions obtained from the algorithm with the true Pareto front as described in Section 8.3. Low values for each metric indicate a higher degree of diversity and closeness to the Pareto front.

From Table 6, it can be observed that the results for WSPSO are comparable with WSGA for the area, delay, and power values whereas WSPSO exhibits an average improvement of 26.19% in the execution times. Thus there is scope for improvement in the WSPSO methodology by enhancing the search method without an adverse impact on execution times. However WSPSO, as in the case of WSGA, fares poorer than NSGA II in terms of solution spread, diversity, closeness to Pareto front, and execution times.

Average measurements are not indicative of the ability of the algorithm to search the entire multi-objective solution space. The diversity of solutions and optimality in terms of closeness to the true Pareto front are also indicators of algorithm efficiency. The performance metrics [6] described in Section 8.2 evaluated for WSGA as well as WSPSO are listed in Table 7. The error ratio is the same for both algorithms. WSPSO fares slightly better in yielding uniformly spaced solutions with better diversity as evidenced by the lower values of *spacing* and *spread* for WSPSO. However, the *generational distance* metric is higher for WSPSO. Thus a higher *generational distance* metric coupled with a lower *MFE* indicates that Rank I individuals have more or less uniform separation from the true Pareto front. This is corroborated by the lower values for spacing and spread mentioned earlier.

9. Conclusions and Future Work

A framework for applying multi-objective evolutionary techniques for multi-objective power, area, and delay optimization for the datapath scheduling problem in High Level Synthesis is presented. The methodology has been applied to the NSGA II algorithm and considerable improvement in runtimes and solution quality is demonstrated compared to a Weighted Sum GA approach by evaluating the technique on standard DFG benchmarks. Thus the technique will be effective as a tool for design space exploration during DFG synthesis. The methodology has been extended to Weighted Sum PSO and preliminary results indicate that WSPSO is faster than WSGA with potential for improvement in solution convergence and quality. The following conclusions can be drawn from the analysis of the WSGA, NSGA II, and WSPSO algorithms.

(i) NSGA II exhibits the best solution quality among all the techniques analyzed and fastest execution times and emerges as the best approach among the three methods compared from the point of view of efficient design space exploration. This is in line with expectations since NSGA II incorporates features for ensuring diversity and convergence to the Pareto front. The method for identifying non-dominated solutions is computationally fast thus resulting in the reduced runtime.

(ii) WSPSO is computationally simple and exhibits lesser run time compared to WSGA since the evolution process does not involve crossover and mutation which are computation intensive.

(iii) However, WSPSO may not be an alternative to NSGA II unless the solution quality is improved by incorporating some hybrid approaches like interweaving with Simulated Annealing as reported in [16]. This aspect needs to be investigated and will be part of the future work.

(iv) True Multi-objective PSO (MOPSO) techniques have been proposed such as the works reported in [9, 11]. This will be adapted for the HLS scheduling problem and the performance vis-à-vis NSGA II will be studied.

(v) The solutions obtained by WSPSO, WSGA, and NSGA II will be synthesized to a target library and the trends predicted by the DFG metrics used in the cost functions with actual estimates from postbinding synthesis results will be compared.

References

[1] D. S. Harish Ram, M. C. Bhuvaneswari, and S. M. Logesh, "A novel evolutionary technique for multi-objective power, area and delay optimization in high level synthesis of datapaths ," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI '11)*, pp. 290–295, Chennai, India, July 2011.

[2] V. Krishnan and S. Katkoori, "A genetic algorithm for the design space exploration of datapaths during high-level synthesis," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 3, pp. 213–229, 2006.

[3] F. Ferrandi, P. L. Lanzi, D. Loiacono, C. Pilato, and D. Sciuto, "A multi-objective genetic algorithm for design space exploration in high-level synthesis," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI '08)*, pp. 417–422, Montpellier, France, April 2008.

[4] E. Kursun, R. Mukherjee, and S. Ogrenci Memik, "Early quality assessment for low power behavioral synthesis," *Journal of Low Power Electronics*, vol. 1, no. 3, pp. 1–13, 2005.

[5] Mediabench benchmark, express.ece.ucsb.edu/benchmark/.

[6] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*, John Wiley and Sons, 2003.

[7] K. Deb, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[8] M. Reyes-Sierra and C. A. Coello Coello, "Multi-objective particle swarm optimizers: a survey of the state-of-the-art," *International Journal of Computational Intelligence Research*, vol. 2, no. 3, pp. 287–308, 2006.

[9] E. Zitzler and L. Thiele, "Multiobjective optimization using evolutionary algorithms—a comparative case study," in *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, pp. 292–304, Springer, London, UK, 1998.

[10] M. S. Bright and T. Arslan, "Multi-objective design strategy for high-level low power design of DSP systems," in *Proceedings of the ISCAS '99*, June 1999.

[11] N. Padhye, J. Branke, and S. Mostaghim, "Comprehensive comparison of MOPSO methods: study of convergence and diversity—survey of state of the art," Tech. Rep., CEC, 2009.

[12] K. P. Wang, L. Huang, C. G. Zhou, and W. Pang, "Particle swarm optimization for traveling salesman problem," in *Proceedings of the 2nd International Conference on Machine Learning and Cybernetics*, November 2003.

[13] D. Lin, S. Qiu, and D. Wang, "Particle swarm optimization based on neighborhood encoding for traveling salesman problem," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, October 2008.

[14] J. Jie, H. Ji, M. Wang, and M. Zhao, "Improved discrete particle swarm optimization based on edge coding and multilevel reduction strategy for larger scale TSP," in *Proceedings of the 6th International Conference on Natural Computation (ICNC '2010)*, August 2010.

[15] R. F. Abdel-Kader, "Particle swarm optimization for constrained instruction scheduling," *VLSI Design*, vol. 2008, Article ID 930610, 7 pages, 2008.

[16] L. Fang, P. Chen, and S. Liu, "Particle swarm optimization with simulated annealing for TSP," in *Proceedings of the 6th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases*, Corfu Island, Greece, February 2007.

[17] S. H. Gerez, *Algorithms for VLSI Design Automation*, John Wiley and Sons, 2000.

[18] C. G. Lyuh and T. Kim, "High-level synthesis for low power based on network flow method," *IEEE Transactions on VLSI Systems*, vol. 11, no. 3, pp. 364–375, 2003.

[19] A. Davoodi and A. Srivastava, "Effective techniques for the generalized low-power binding problem," *ACM Transactions on Design Automation of Electronic Systems*, vol. 11, no. 1, pp. 52–69, 2006.

[20] X. Tang, T. Jiang, A. Jones, and P. Banerjee, "Behavioral synthesis of data-dominated circuits for minimal energy implementation," in *Proceedings of the International Conference on VLSI Design: Power Aware Design of VLSI Systems*, January 2005.

[21] N. Chabini and W. Wolf, "Unification of scheduling, binding, and retiming to reduce power consumption under timings and resources constraints," *IEEE Transactions on VLSI Systems*, vol. 13, no. 10, pp. 1113–1126, 2005.

[22] A. K. Murugavel and N. Ranganathan, "A game theoretic approach for power optimization during behavioral synthesis," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 6, pp. 1031–1043, 2003.

[23] C. Mandal, P. P. Chakrabarti, and S. Ghose, "GABIND: a GA approach to allocation and binding for the high-level synthesis of data paths," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 6, pp. 747–750, 2000.

[24] J.-M. Chang and M. Pedram, "Register allocation and binding for low power," in *Proceedings of the Design Automation Conference (DAC '95)*, June 1995.

Hindawi Publishing Corporation
VLSI Design
Volume 2012, Article ID 589128, 9 pages
doi:10.1155/2012/589128

Research Article

Line Search-Based Inverse Lithography Technique for Mask Design

Xin Zhao and Chris Chu

Department of Electrical and Computer Engineering, Iowa State University, Ames, IA 50011, USA

Correspondence should be addressed to Xin Zhao, xinzhao@iastate.edu

Received 24 March 2012; Accepted 3 August 2012

Academic Editor: Gi-Joon Nam

Copyright © 2012 X. Zhao and C. Chu. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

As feature size is much smaller than the wavelength of illumination source of lithography equipments, resolution enhancement technology (RET) has been increasingly relied upon to minimize image distortions. In advanced process nodes, pixelated mask becomes essential for RET to achieve an acceptable resolution. In this paper, we investigate the problem of pixelated binary mask design in a partially coherent imaging system. Similar to previous approaches, the mask design problem is formulated as a nonlinear program and is solved by gradient-based search. Our contributions are four novel techniques to achieve significantly better image quality. First, to transform the original bound-constrained formulation to an unconstrained optimization problem, we propose a new noncyclic transformation of mask variables to replace the wellknown cyclic one. As our transformation is monotonic, it enables a better control in flipping pixels. Second, based on this new transformation, we propose a highly efficient line search-based heuristic technique to solve the resulting unconstrained optimization. Third, to simplify the optimization, instead of using discretization regularization penalty technique, we directly round the optimized gray mask into binary mask for pattern error evaluation. Forth, we introduce a jump technique in order to jump out of local minimum and continue the search.

1. Introduction

For modern very large-scale integration (VLSI) design, the traditional VLSI physical design problems (e.g., floorplanning [1–3], clustering [4, 5], placement [6], and routing) used to play the critical role on coping with the ever-increasing design complexity. However, as semiconductor manufacturers move to advanced process nodes (especially 45 nm process and below), lithography has become a greater challenge due to the fundamental constraints of optical physics. Because feature size is much smaller than the wavelength of illumination source (currently 193 nm), the image formed on wafer surface is distorted more and more seriously due to optical diffraction and interference phenomena. The industry has been investigating various alternatives (e.g., EUV lithography, E-beam lithography), but none of them is ready in the foreseeable future. As a result, semiconductor manufacturers have no choice but to keep using the existing equipments in patterning the progressively smaller features.

Given the limitation of lithography equipments, resolution enhancement technology (RET) such as optical proximity correction (OPC), phase shift mask (PSM), and double patterning has been increasingly relied upon to minimize image distortions [7]. In recent years, pixelated mask, which allows great flexibility in the mask pattern, has become essential for RET to achieve better resolution.

For the design of pixelated mask, the most popular and successful approach is to formulate it as a mathematical program and solve it by gradient-based search [8–14]. Granik [8] considered a constrained nonlinear formulation. Poonawala and Milanfar [9, 14, 15] proposed an unconstrained nonlinear formulation, and employed a regularization framework to control the tone and complexity of the synthesized masks. Ma and Arce [11, 16] presented a similar unconstrained nonlinear formulation targeting PSM. Ma and Arce [12, 16] focused on partially coherent illumination and used singular value decomposition to expand the partially coherent imaging equation by eigenfunctions into a sum of coherent systems (SOCSs). All works discussed above

utilized the steepest descent method to solve the nonlinear programs. Ma and Arce [10] demonstrated that the conjugate gradient method is more efficient. The work of Yu and Pan [17] is an exception to the mathematical programming approach Instead, a model-based pixel flipping heuristic is proposed.

In this paper, we focus on the design of pixelated binary mask in a partially coherent imaging system (the techniques proposed in this paper can all be easily extended to PSM and other imaging systems). Similar to previous approaches, we formulate the problem as an unconstrained nonlinear program and solve it by iterative gradient-based search. The main contributions of this paper are listed below.

(i) To transform the problem formulation from a bounded optimization to an unconstrained one, we propose a new noncyclic transformation of mask variables to replace the widely used cyclic one. Our transformation is monotonic and allows a better control of flipping pixels.

(ii) Based on this new transformation, we present a highly efficient line search-based technique to solve the resulting unconstrained optimization. Because of the non-cyclic nature of the transformation, the solution space is not so rugged. Therefore, our algorithm can find much better binary masks for the inverse lithography problem.

(iii) A jump technique: as gradient-based search techniques will be trapped at a local minimum, we introduce a new technique named *jump* in order to jump out of the local minimum and continue the search.

(iv) We apply a direct rounding technique to regularize gray masks into binary ones instead of adding a discretization regularization penalty to the cost function as in [14] and [16]. This simplifies the computation and achieves better results as the experiment results show.

The rest of this paper is organized as follows. The formulation of the inverse lithography problem is explained in Section 3. Section 4 describes in details the flow of our algorithm and the four novel techniques that we proposed. Section 5 presents the experimental results. The paper is concluded in Section 6.

2. New Algorithmic Technique Used

The inverse lithography technique for mask design has been proposed in [8, 15] in 2006 and has been widely discussed in recent years as semiconductor manufacturers move to advanced process nodes. But so far, there is not an effective search method proposed because of the complicated solution space of this problem. We introduce a novel transformation for mask pixel, which enables an effective line search technique.

3. Problem Formulation

In an optical lithography system, a photo mask is projected to a silicon wafer through an optical lens. An aerial image of the mask is then formed on the wafer surface, which is covered by photoresist. After developing and etching, a pattern similar to the one on the mask is formed on the wafer surface. To simulate the pattern formation on the wafer surface for a given mask, we first describe below a projection optics model and a photoresist model. After that, we present the formulation of the mask design problem.

3.1. Projection Optics Model. The Hopkins diffraction model [13] is widely used to approximate partially coherent optics systems. To reduce the computational complexity of the Hopkins diffraction model, the Fourier series expansion model [18] is a common approach. In this paper, we followed this model.

The Fourier series expansion model approximates the partially coherent imaging system as a sum of coherent system (SOCS). Based on this model, the computation of the aerial image I of a pixelated mask M is given in (1) and illustrated in Figure 1. Here, the dimensions of the pixelated mask and the image are $m \times n$. The illumination source is partitioned into $N \times N$ sources. u and are the Fourier series coefficients and spatial kernels, respectively:

$$I(M) = \sum_{p=1}^{N \cdot N} u_p \left| h_p \otimes M \right|^2. \tag{1}$$

Note that the convolution $h \otimes M$ can be achieved in frequency domain using fast Fourier transform $\mathcal{FFT}$ and inverse fast Fourier transform $\mathcal{FFT}^{-1}$ as shown in the following:

$$I(M) = \sum_{p=1}^{N \cdot N} u_p \left| \mathcal{FFT}^{-1} \left\{ \mathcal{FFT}\left(h_p\right) \cdot \mathcal{FFT}(M) \right\} \right|^2. \tag{2}$$

3.2. Photoresist Model. To model the reaction of the photoresist to the intensity of light projected on it, we use the constant threshold model as follows

$$z_i = \begin{cases} 0 & \text{if } I_i \le t_r, \\ 1 & \text{if } I_i > t_r, \end{cases} \tag{3}$$

where I_i and z_i are the light intensity and the corresponding reaction result of the photoresist at pixel i on the wafer surface, respectively, and t_r is the threshold of the photoresist.

Thus, the pattern z formed on the wafer surface can be expressed as a function of the mask M based on (2) and (3). In order to make z differentiable so that gradient-based search can be applied, we approximate the above constant threshold model with the sigmoid function

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-a(x-v)}}, \tag{4}$$

where the parameter a determines the steepness of the sigmoid function around $x = v$. The larger value of a is,

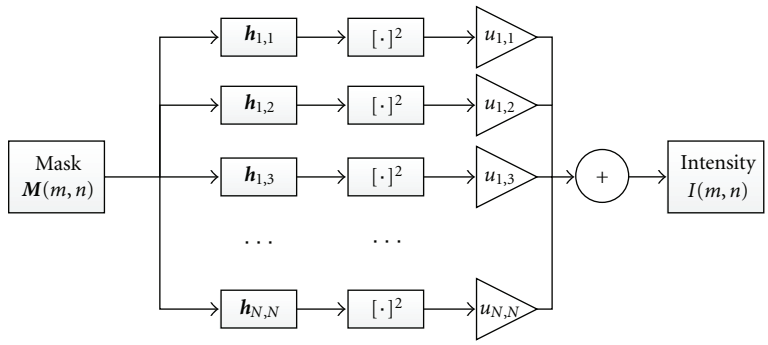


Figure 1: Fourier series expansion model for partially coherent system.

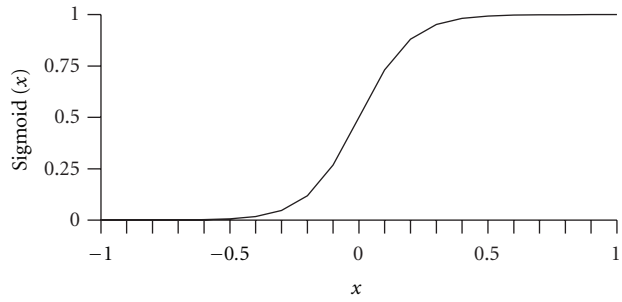


Figure 2: The sigmoid function with $a = 10$ and $v = 0$.

the steeper and hence the closer to the constant threshold model the sigmoid function will be. The sigmoid function with $a = 10$, $v = 0$ is illustrated in Figure 2.

Using the sigmoid function, the reaction of the photoresist at pixel i for a mask M is

$$z_i(M) \approx \frac{1}{1 + e^{-a(I_i(M) - t_r)}}. \tag{5}$$

3.3. Our Inverse Lithography Problem Formulation. Inverse lithography treats mask design as an inverse problem of imaging. Given a target pattern $\hat{z}$, the problem is to find a mask M^* such that the corresponding pattern $z(M^*)$ on the wafer surface is as close to $\hat{z}$ as possible [19].

The error between the target pattern $\hat{z}$ and the generated pattern $z(M)$ for any mask M is commonly defined as

$$E(M) = \sum_{i=1}^{m \cdot n} (\hat{z}_i - z_i(M))^2. \tag{6}$$

So the inverse lithography problem is formulated as

$$M^* = \arg\min_{\forall i, M_i \in \{0,1\}} E(M). \tag{7}$$

Combining (5) and (6) with (7), the problem is written as

$$M^* = \arg\min_{\forall i, M_i \in \{0,1\}} \sum_{i=1}^{m \cdot n} \left(\hat{z}_i - \frac{1}{1 + e^{-a(I_i(M) - t_r)}} \right)^2, \tag{8}$$

where $I_i(M)$ is the light intensity at pixel location i calculated by (2).

4. Line Search-Based Inverse Lithography Technique

As the value of each pixel M_i should be 0 or 1, the inverse lithography problem is an integer nonlinear program. To make it easier to solve, a common approach is to relax the integer constraints to $0 \le M_i \le 1$ for all i [8–12, 14]. Therefore, the problem becomes a bounded non-linear program. To further simplify the program, it is also common to convert it into an unconstrained non-linear program [8–12, 14]. It is achieved by a transformation $M_i = T(\beta_i)$ which maps an unbounded variable β_i into the range $[0, 1]$. (We will discuss this transformation in Section 4.1.) The program is then solved with respect to β's domain.

This unconstrained non-linear program can be solved by an iterative gradient-based search method. Starting from some point β in the solution space, a search direction, which can be decided based on the gradient of (6), is first found. Then a step of a certain size along the search direction is taken. Thus, a new point, which hopefully has less pattern error, is reached. The search is repeated until the error cannot be further reduced.

In this paper, we apply this iterative gradient-based search method, which is outlined in Algorithm 1. Our contributions are four novel techniques as described in Sections 4.1, 4.2, 4.3, and 4.4 to reduce pattern error over previous works.

In particular, we use the steepest descent method, that is, the search direction is the negative of the local gradient of (6). But our techniques are not limited only to the steepest descent method. It should be applicable to other iterative gradient-based search approaches like conjugate gradient method.

4.1. Novel Transformation for Mask Pixel. As explained above, to convert the inverse lithography problem into an unconstrained optimization problem, we need a transformation $T : \mathcal{R} \to [0, 1]$. Then we can use an unbounded variable β_i to represent each pixel based on $M_i = T(\beta_i)$.

One such transformation is proposed by Poonawala and Milanfar [14]:

$$M_i = \frac{1 + \cos\beta_i}{2}. \tag{9}$$

(1) Transform initial mask into β // Section 4.1
(2) Repeat
(3) Find the search direction d at β // Equation (12)
(4) Determine the step size S // Section 4.2
(5) $\beta^{\text{new}} = \beta + S * d$
(6) Generate *gray* mask $M = T(\beta^{\text{new}})$ // Equation (11)
(7) // Round M to binary as described in Section 4.4
(8) Evaluate pattern error $E(M)$ // Equations (2), (3), and (6)
(9) $\beta = \beta^{\text{new}}$
(10) Until pattern error is not improving

Algorithm 1: Generic framework for iterative gradient-based search. Note that specific details about our algorithm are given in the comments.

This idea is widely adopted by later works [9–12]. We call it the cosine transformation.

In gradient-based search, a line search along the search direction is typically performed to determine the step size S to get to a local minimum (step 4 in Algorithm 1). The line search will be more effective if the function $E(M)$ along the search direction is smooth and, better yet, convex. However, the cosine transformation is a cyclic function. It is clearly not a one-to-one transformation. By increasing the value of β_i, M_i changes its value between 0 and 1 periodically. As a result, when β is moving along any direction, E(M) may keep jumping up and down as M_i keeps switching between 0 and 1.

To illustrate this, we consider the algorithm described in Chapter 7 of Ma and Arce [16], which solved the same problem formulation as our paper. It also applied the steepest descent method, but it used the cosine transformation. The pattern error function (6) is turned into the following:

$$E(\beta) = \sum_{i=1}^{m \cdot n} \left(\widehat{z}_i - \frac{1}{1 + e^{-a(\sum_{p=1}^{N \cdot N} u_p |h_p \otimes ((1+\cos\beta)/2)|^2 - t_r)}} \right)^2. \tag{10}$$

Using the software code and the target pattern (as shown in Figure 3) provided by [16], when β is moved along the negative gradient direction of (10), the function $E(\beta)$ is illustrated in Figures 4 and 5. It shows that the function changes in a very chaotic manner. We have observed a large number of experiments on different target patterns and different current masks. The function $E(\beta)$ always shows a similar chaotic behavior. It makes line search very difficult. In theory, the negative gradient points out the direction for each pixel to be adjusted to achieve the minimal value of pattern error. However, the gradient only provides the direction of change at the local point. Because of the cyclic property of (9), the pixels on the mask may be flipped to the wrong direction if the step size S is not set appropriately. This makes the gradient-based search method very ineffective. In fact, the common practice in previous works [9–12, 14] is to set the step size to some fine-tuned constant instead of computed-by-line search.

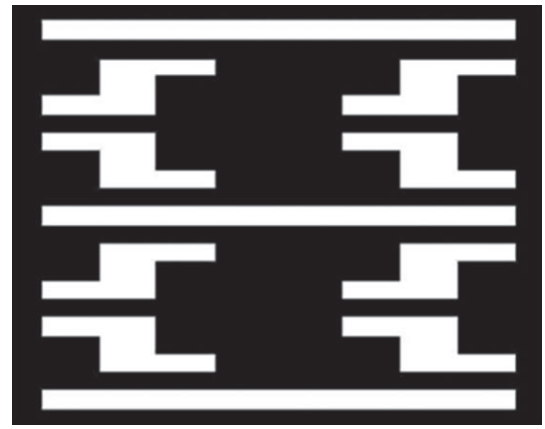


Figure 3: Target pattern from [16] with 184×184 pixels.

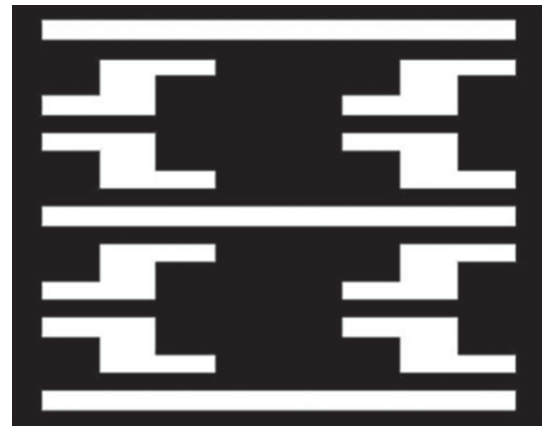


Figure 4: Pattern error based on cosine transformation.

We propose a new transformation for M_i based on the sigmoid function (see (4)):

$$M_i = \frac{1}{1 + e^{-A(\beta_i - T_R)}}, \tag{11}$$

where A is the steepness control parameter and T_R specifies the transition point of the function. A larger A will cause the pixel values to be closer to 0 or 1. T_R can be set to any value and is set to 0 in this paper. We call this the sigmoid

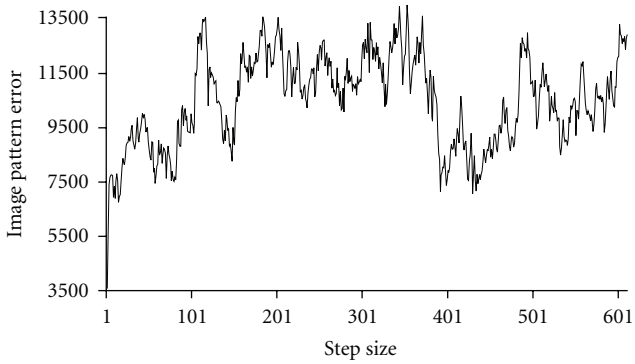


Figure 5: Enlarged version of Figure 4 with step size from = 0 to 25.

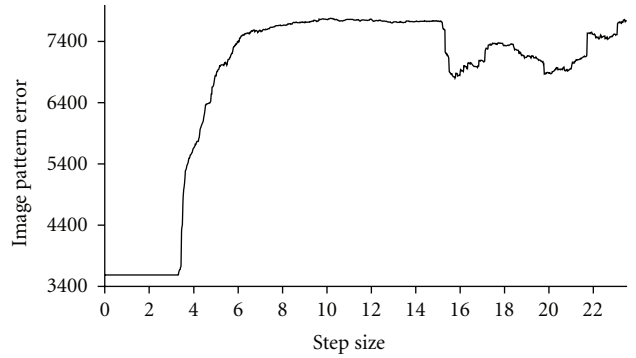


Figure 6: Pattern error based on sigmoid transformation.

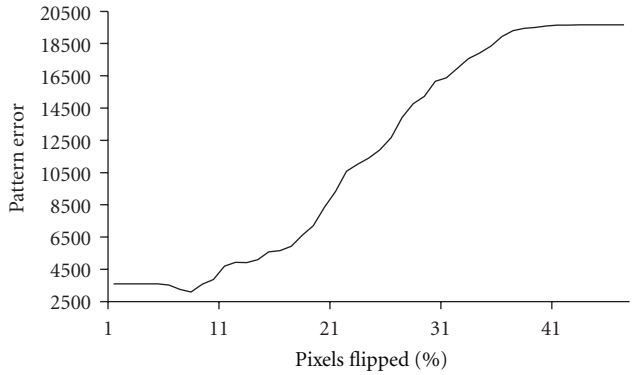


Figure 7: Enlarged version of Figure 6 with percentage from 5.1% to 9.0%.

transformation. As the sigmoid transformation is a strictly increasing function, when β is moved along any direction, each mask pixel is flipped at most once.

Based on the sigmoid transformation, the gradient of (6) is

$$\begin{aligned}\frac{\partial E(\beta)}{\partial \beta} = & -a \cdot A\left\{\sum_{p=1}^{N\cdot N} u_p\Big[\Big(h_p \otimes M\Big) \odot (\hat{z}-z) \odot z \right. \\ & \left. \odot(1-z)\Big] \otimes h_p^{*T}\right\} \\ & \odot M \odot (1-M) - a \cdot A \\ & \cdot\left\{\sum_{p=1}^{N\cdot N} u_p\Big[\Big(h_p^{*} \otimes M\Big) \odot (\hat{z}-z) \odot z \right. \\ & \left. \odot(1-z)\Big] \otimes h_p^{T}\right\} \\ & \odot M \odot (1-M),\end{aligned} \tag{12}$$

where $1 = [1,\ldots,1]^T \in \mathcal{R}^{m\times n}$, $\odot$ is the element-by-element multiplication operator, and h_p^* is the conjugate of h_p. We have performed a large number of experiments on different target patterns and different current masks. When β is moved along the negative gradient direction, the function $E(\beta)$ is almost always unimodal. One typical example is shown in Figures 6 and 7. (Note that not every pixel can be flipped along the negative gradient direction, as we will explain in Section 4.2.) This makes it feasible to apply line search to heuristically minimize the pattern error. Note that gradient calculation is very expensive due to the four convolutions in (12). Hence, once a gradient is calculated, it is desirable to perform line search to minimize the pattern error as much as possible in order to reduce the number of iterations (i.e., gradient calculations) of the gradient-based search algorithm. As Figure 7 shows, by performing line search along the negative gradient direction, the image pattern error can be effectively reduced from around 3600 to below 3100 in one iteration.

4.2. Highly Efficient Line Search Technique. In this section, we present a highly efficient line search technique to determine the step size in step 4 in Algorithm 1 to minimize pattern error. We observe that in each iteration, the shape of the function $E(\beta)$ along the direction of the negative gradient is almost always like the curve shown in Figure 6. We employ golden section method for line search. Golden section search is an iterative technique which successively narrows the search range.

Because the final optimized mask should be a binary one, we need to round the gray mask, which is given by (11), to binary according to some rounding threshold t_m. In other words,

$$M_i^{\text{binary}}(\beta_i) = \begin{cases} 0, & M_i(\beta_i) < t_m, \\ 1, & M_i(\beta_i) \geq t_m, \end{cases} \tag{13}$$

where M^{binary} is the resulting binary mask. Here, we simply set t_m to 0.5.

When moving along the negative gradient direction, as the value of each pixel M_i is changed monotonically due to our new transformation, we can easily control the number of pixels flipped (i.e., changed from below t_m to above t_m or vice versa) during line search. This idea is explained below.

Given the current mask specified by β and the negative gradient direction d, (11) can be written as a function of S as

$$M_i(S) = \frac{1}{1 + e^{-A(\beta_i - S \times d_i - T_R)}}. \quad (14)$$

By substituting (14) into (13) and rearranging, we get the following

if $d_i \geq 0$,

$$M_i^{\text{binary}}(S) = \begin{cases} 0, & S > S_i, \\ 1, & S \leq S_i, \end{cases} \quad (15)$$

if $d_i < 0$,

$$M_i^{\text{binary}}(S) = \begin{cases} 0, & S < S_i, \\ 1, & S \geq S_i, \end{cases} \quad (16)$$

where

$$S_i = \frac{\beta_i - (lg(1/t_m - 1)/ - A) - T_R}{d_i} \quad (17)$$

is the threshold on step size for flipping pixel i. At the current mask, if a pixel's value is less than t_m and its negative gradient is positive, or if a pixel's value is larger than t_m and its negative gradient is negative, then the pixel will be flipped when we apply a step size S larger than S_i. Other pixels are unflippable no matter how large step size S we use. So it is easy to determine how many pixels can be flipped. To control the number of pixels flipped during golden section search, we first mark all flippable pixels along the negative gradient direction. Then we calculate the threshold on step size, S_i, for each flippable pixel i. By sorting these thresholds from smallest to largest, the number of pixels flipped can be controlled by setting the value of S. For example, by using the 50th value of the sorted thresholds as the step size S, 50 pixels will be flipped along the negative gradient. In golden section search, the minimum and maximum sorted thresholds can be used to define the search region. In this paper, we use a segment in the region from the minimum to the maximum sorted thresholds as our search region. The details will be discussed in Section 5.

4.3. Jump Technique. Because of the noncyclic nature of our transformation, the solution space is not so rugged. But it is still extremely complicated with many local minima. As gradient-based search techniques will be trapped at a local minimum, we introduce a new technique named *jump* in order to jump out of the local minimum and continue the search. During the line search process, if the algorithm cannot find a better solution along the search direction (i.e., gets trapped at some local minimum), instead of terminating, it will jump along the search direction with a large step size to a probably worse solution. Then the algorithm will continue the gradient-based search starting from the new solution. If the step size is large enough, it is likely that the algorithm will not converge to the previous local minimum. At the end, the algorithm will return the best local minimum that has been found. For example, if 2 jumps happened, there would be 3 local minima, the first one was found without jump, and the other 2 were found by 2 jumps. Our program keeps recording the local minima and returns the best one at last.

4.4. Direct Rounding of Gray Mask. In order to apply gradient-based approach, it is unavoidable to relax the integer constraints. As a result, the optimized mask becomes a gray one. Because our goal is to generate a binary mask, the optimized gray mask has to be rounded to a binary one at last. A regularization framework was proposed in [14, Section IV.A] and also in [16, Chapter 6.1] to bias the output gray mask to be closer to binary. This regularization approach adds to the objective function (i.e., (6)) a quadratic penalty term for each pixel. However, it is still hard to control the change in the image pattern error caused by the rounding of the gray mask at the end. The optimized gray mask may achieve a low pattern error. However, after rounding the gray mask into binary, the pattern error often increases dramatically. Instead of using the quadratic penalty regularization framework, we propose to directly round the optimized gray mask into a binary one in each iteration before evaluating the pattern error. In this way, we simplify the objective function and also guarantee that our search will not be misled by inaccurate pattern error values. we observed that it works well based on our experiments.

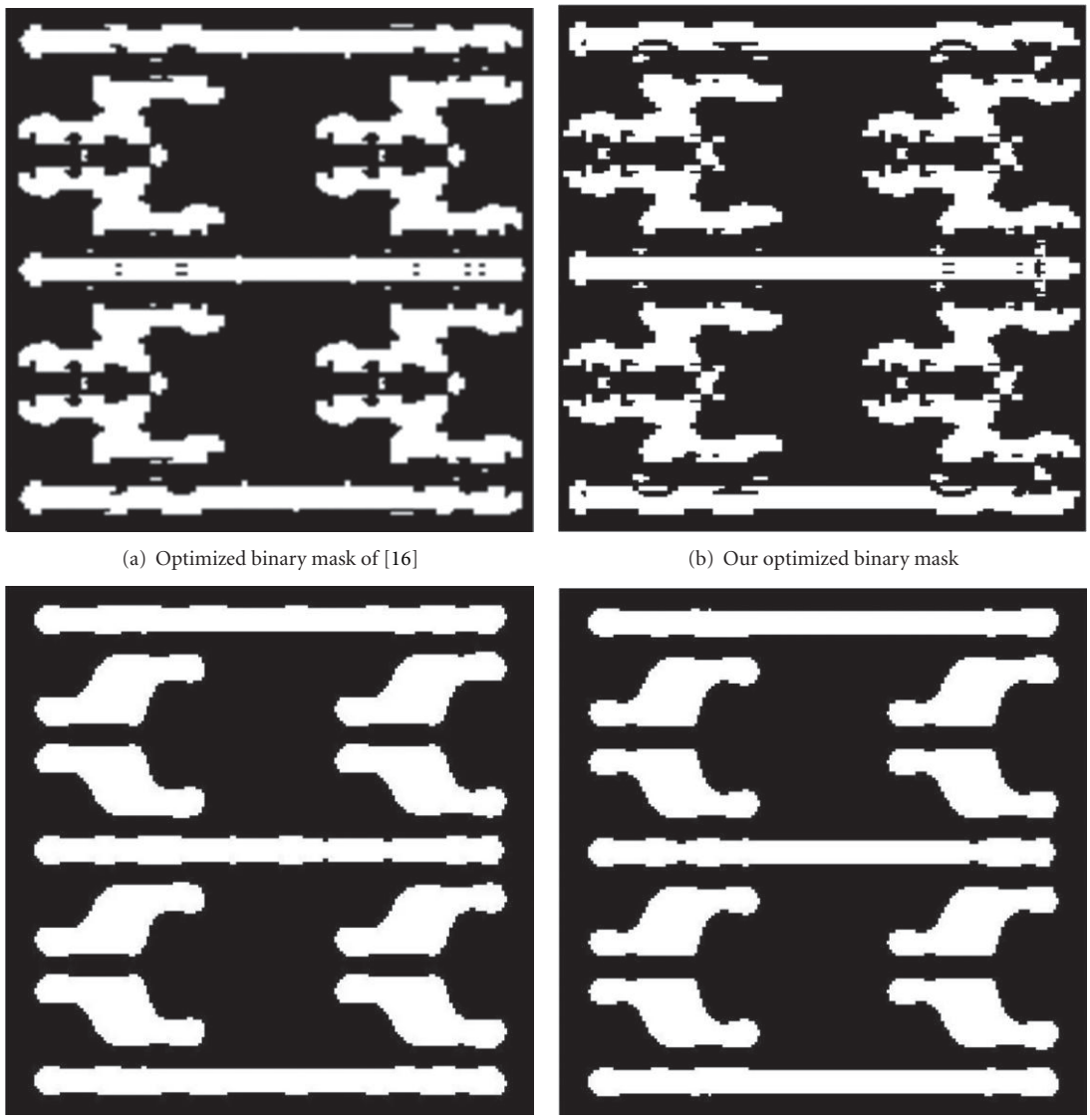
5. Experimental Results

We compare an implementation of our algorithm with the program developed by Ma and Arce [16]. Both of the programs are coded in Matlab and executed on an Intel Xeon(R) X5650 2.67 GHz CPU. The Matlab program of Ma and Arce [16] is public, and we downloaded it from the publisher. The runtime reported is CPU time, and the programs are restricted to use a single core when running in Matlab.

In [16], the program uses cosine transformation and a preset step size of 2. Besides, it applies regularization with a quadratic penalty term as mentioned in Section 4.4. Isolated perturbations, protrusions, and so forth are very hard to be written by the mask writer, so in [16], it also applies another regularization called complexity penalty term, which restricts the complexity of optimized binary mask. The details can be found in [14, Section IV.B] and also in [16, Chapter 6.2]. To have a fair comparison, we followed previous works and our program applies the complexity penalty regularization too. But as we mentioned in Section 4.4, our program does not apply the quadratic penalty regularization but directly rounds the optimized gray mask into a binary one instead whenever pattern error is evaluated. In [16], the stopping criteria of gradient-based search is set in an ad hoc manner according to the target pattern. In order to fairly compare the two programs on various masks, the same stopping criteria are applied to both programs. The criteria are that if the average pattern error over the last 30 iterations is larger than the average pattern error over the 30 iterations before that,

Table 1: Pattern and runtime comparison between [16] and ours.

No.	Mask size (pixel)	Pixel size (nm)	Feature size (nm)	Pattern error					Runtime (s)		
				[16]	(%)	[16] with our runtime	(%)	Ours	[16]	(%)	Ours
1	184 × 184	5.625	45	1512	(9.88)	1566	(13.81)	1376	192	(−49.61)	381
2	400 × 400	5.625	45	3308	(24.17)	3780	(41.89)	2664	13337	(13.24)	11778
3	2000 × 2000	5.625	45	108144	(8.55)	109267	(9.68)	99624	17918	(−0.38)	17986
4	2000 × 2000	5.625	45	61350	(10.72)	66516	(20.05)	55409	12457	(19.81)	10397
5	4000 × 4000	5.625	45	58410	(198.39)	58410	(198.39)	19575	78652	(16.34)	67603
6	2000 × 2000	4	32	101785	(75.18)	102100	(75.72)	58104	49333	(13.45)	43485
7	2000 × 2000	4	32	64252	(39.49)	75300	(63.47)	46063	51856	(−21.57)	66117
8	4000 × 4000	4	32	148356	(358.80)	148420	(358.99)	32336	62008	(43.35)	43255
9	4000 × 4000	4	32	52160	(153.30)	52160	(153.30)	20592	71489	(5.77)	67588
	Average				97.61		103.92			4.49	



(a) Optimized binary mask of [16]

(b) Our optimized binary mask

(c) Image pattern of optimized binary mask of [16], error = 1512

(d) Image pattern of our optimized binary mask, error = 1376

Figure 8: Comparison of optimized binary mask and pattern error for target pattern no. 1.

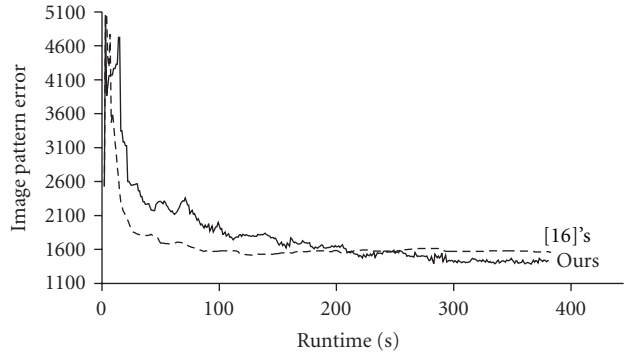


Figure 9: The convergence curve for target pattern no. 1.

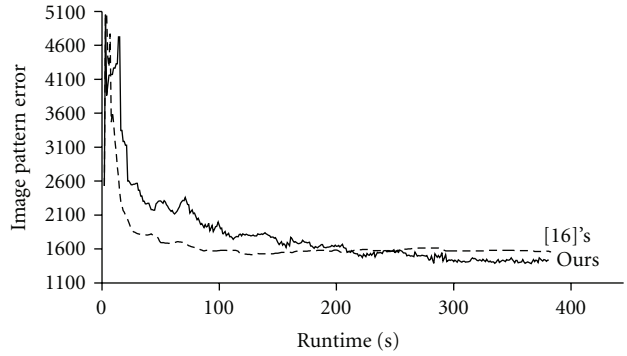


Figure 10: The convergence curve for target pattern no. 6.

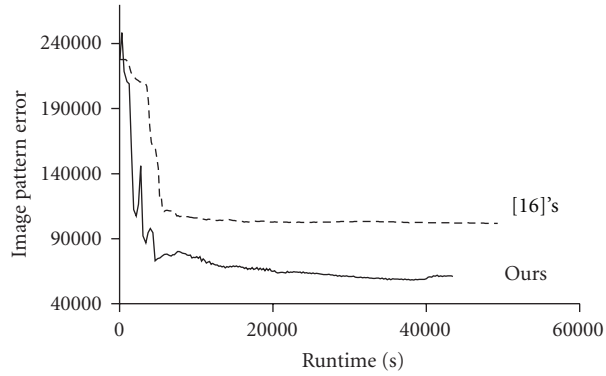


Figure 11: The convergence curve for target pattern no. 7.

the program will stop. For evaluation of pattern error in both programs, in each iteration, the optimized gray mask is rounded using (13). We use the same convolution kernel h as the Matlab program of [16].

For the photoresist model, we use $a = 25$ and $t_r = 0.19$ for the sigmoid function in (5). For the transformation of mask variables from β, we use $A = 4$ and $T_R = 0$ for the sigmoid function in (11). The threshold t_m in (13) is set to 0.5.

Based on our observation of many experiments, for the first iteration of gradient-based search, the minimum pattern error can almost always be achieved by flipping less than 10% of all pixels. One example is showed in Figures 6 and 7, where the minimum pattern error is at about 7.7%. Then in the later iterations, this region remains nearly the same or keeps shrinking. So for the first 2 iterations, we set the initial search region of golden section search to be the region in which the first 10% of overall pixels can be flipped along the negative gradient direction. Our program keeps recording the minimum location which is found in each iteration to guide the search region for the next iteration. For example, if in the current iteration, the minimum error is found at 5% of the overall pixels flipped, to be on the safe side, the search region of the next iteration will be automatically set as 1.5 times of 5%, which is 7.5%, of the pixels flipped. To prevent this search region from shrinking too small, we set a minimum as 2%. For the stop criteria of the golden section search, we set it as 0.25%, which means that when the search region shrinks to or below 0.25%, our program will stop searching. As mentioned above for the jump technique, if our program cannot find a better solution along the search direction until it stops searching (i.e., gets trapped at some local minimum), our program will take the best solution, except the starting point of that line search, as a new solution, although it is a worse solution. This means one jump.

The comparisons of pattern error of optimized binary masks and runtime between our program and that of [16] are shown in Table 1. We use 9 binary image patterns as predefined targets. The outer and inner partial coherence factors for 184×184 target pattern are set to 0.4 and 0.3, respectively; for 400×400 target pattern, are set to 0.975 and 0.8, respectively; for all 2000×2000 target patterns, are set to 0.3 and 0.299, respectively; and for all 4000×4000 target patterns, are set to 0.2 and 0.1995, respectively.

The pattern errors reported are calculated according to the best binary mask generated for both programs. All gradient-based methods strongly depend on a starting solution. We followed the previous works and used the target as the starting point to search. The runtimes listed in the last two columns are based on the stopping criteria mentioned above. As the table shows, our program always generates better optimized binary mask which has significantly less pattern error. The pattern errors of [16] are higher than ours by 8.55% to 358.80%, with an average of 97.61%. Moreover, the programn of [16] uses 4.49% more runtime than our program on average.

We report the pattern error of the final binary mask generated for the program of [16] with our program's runtime in column 6 of the table. For target patterns no. 1, no. 3 and no. 7, the program of [16] stops earlier than ours. To see if the program of [16] will converge to better solutions if more runtime is allowed, we change the stopping criteria to let it run until the same runtime as that of our program. The result shows that the error gets worse in all 3 cases. If the pattern error of the best binary mask generated is reported instead, the result will be exactly the same as in column 5. It indicates that the program fails to get out of the local minima even with more time.

Target pattern no. 1 is obtained from [16]. We illustrate the optimized binary masks and the corresponding image

patterns for both programs in Figure 8. The pattern error convergence curves are shown in Figure 9.

More pattern error convergence curves are shown in Figures 10 and 11 for target patterns no. 6 and no. 7, respectively.

Because the program we obtained from [16] is fine-tuned for the target pattern no. 1 which is also obtained from [16], the experiment result of our algorithm is not so much better than that of [16]. However, based on the experiment results of other target patterns which cover multiple mask sizes, pixel sizes, and feature sizes, our algorithm has an overwhelming advantage due to the application of line search engine which is enabled by our novel transformation of mask pixel. Based on the observation of Figures 10 and 11, the program of [16] is very easy to be trapped because line search is not applied and a fixed step size is used. On the other hand, benefited from line search and jump technique, our program has better performance. Even if our program is trapped, the jump technique enables the algorithm to jump out and continue the search.

6. Conclusion

In this paper, we introduced a highly efficient gradient-based search technique to solve the inverse lithography problem. We proposed a new noncyclic transformation of mask variables to replace the well-known cyclic one. Our transformation is monotonic, and it enables a much better control in flipping pixels and the use of line search to minimize the pattern error. We introduced a new technique named jump in order to jump out of the local minimum and continue the search. We used direct rounding technique to simplify the optimization. The experimental results showed that our technique is significantly more effective than the state of the art. It produces better binary masks in a similar runtime. The four techniques we proposed should be applicable to other iterative gradient-based search approaches, like the conjugate gradient method. We plan to incorporate our techniques into other search methods in the future.

References

[1] J. Z. Yan and C. Chu, "DeFer: deferred decision making enabled fixed-outline floorplanner," in *Proceedings of the 45th Design Automation Conference (DAC '08)*, pp. 161–166, June 2008.

[2] J. Z. Yan and C. Chu, "DeFer: deferred decision making enabled fixed-outline floorplanning algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 3, pp. 367–381, 2010.

[3] J. Z. Yan and C. Chu, "Optimal slack-driven block shaping algorithm in fixed-outline floorplanning," in *Proceedings of the ACM International Symposium on Physical Design (ISPD '12)*, pp. 179–186, 2012.

[4] J. Z. Yan, C. Chu, and W. K. Mak, "SafeChoice: a novel clustering algorithm for wirelength-driven placement," in *Proceedings of the ACM International Symposium on Physical Design (ISPD '10)*, pp. 185–192, March 2010.

[5] J. Z. Yan, C. Chu, and W. K. Mak, "SafeChoice: a novel approach to hypergraph clustering for wirelength-driven placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 7, pp. 1020–1033, 2011.

[6] J. Z. Yan, N. Viswanathan, and C. Chu, "Handling complexities in modern large-scale mixed-size placement," in *Proceedings of the 46th ACM/IEEE Design Automation Conference (DAC '09)*, pp. 436–441, July 2009.

[7] A. K. Wong, *Resolution Enhancement Techniques in Optical Lithography*, SPIE Press, 2001.

[8] Y. Granik, "Fast pixel-based mask optimization for inverse lithography," *Journal of Microlithography, Microfabrication and Microsystems*, vol. 5, no. 4, Article ID 043002, 2006.

[9] A. Poonawala and P. Milanfar, "A pixel-based regularization approach to inverse lithography," *Microelectronic Engineering*, vol. 84, no. 12, pp. 2837–2852, 2007.

[10] X. Ma and G. R. Arce, "Pixel-based OPC optimization based on conjugate gradients," *Optics Express*, vol. 19, no. 3, pp. 2165–2180, 2011.

[11] X. Ma and G. R. Arce, "Generalized inverse lithography methods for phase-shifting mask design," *Optics Express*, vol. 15, no. 23, pp. 15066–15079, 2007.

[12] X. Ma and G. R. Arce, "PSM design for inverse lithography with partially coherent illumination," *Optics Express*, vol. 16, no. 24, pp. 20126–20141, 2008.

[13] M. Born and E. Wolf, *Principles of Optics*, Cambridge University Press, New York, NY, USA, 1999.

[14] A. Poonawala and P. Milanfar, "Mask design for optical microlithography—an inverse imaging problem," *IEEE Transactions on Image Processing*, vol. 16, no. 3, pp. 774–788, 2007.

[15] A. Poonawala and P. Milanfar, "OPC and PSM design using inverse lithography: a non-linear optimization approach," in *Optical Microlithography XIX*, vol. 6514 of *Proceedings of SPIE*, pp. 1159–1172, February 2006.

[16] X. Ma and G. R. Arce, *Computational Lithography*, John Wiley & Sons, New York, NY, USA, 2010.

[17] P. Yu and D. Z. Pan, "TIP-OPC: a new topological invariant paradigm for pixel based optical proximity correction," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '07)*, pp. 847–853, November 2007.

[18] B. E. A. Saleh and M. Rabbani, "Simulation of partially coherent imagery in the space and frequency domains and by modal expansion," *Applied Optics*, vol. 21, no. 15, pp. 2770–2777, 1982.

[19] L. Pang, Y. Liu, and D. Abrams, "Inverse Lithography Technology (ILT), What is the impact to photomask industry?" in *Proceedings of the Photomask and Next-Generation Lithography Mask Technology XIII*, vol. 6283 of *Proceedings of SPIE*, April 2006.