Integration, the VLSI Journal xxx (xxxx) xxx

Contents lists available at ScienceDirect

Integration, the VLSI Journal

journal homepage: www.elsevier.com/locate/vlsi

Enhancing sensitivity-based power reduction for an industry IC design context

Hamed Fatemi [a], Andrew B. Kahng [b,c], Hyein Lee [c,d,*], Jiajia Li [e], Jose Pineda de Gyvez [a]

[a] *NXP Semiconductors, High Tech Campus 46, 5656 AE Eindhoven, Netherlands*
[b] *CSE Department, University of California at San Diego, La Jolla, California, USA*
[c] *ECE Department, University of California at San Diego, La Jolla, California, USA*
[d] *Synopsys, Inc., Sunnyvale, CA, USA*
[e] *Qualcomm Technologies, Inc., San Diego, CA, USA*

ARTICLE INFO

Keywords:
Gate sizing
Post-route optimization
Physical optimization
Power and timing optimization
Multi-corner multi-mode
Industrial design

ABSTRACT

For many years, discrete gate sizing has been widely used for timing and power optimization in VLSI designs. The importance of gate sizing optimization has been emphasized by academia for many years, especially since the 2012/2013 ISPD gate sizing contests [1, 2]. These contests have provided practical impetus to academic sizers through the use of realistic constraints and benchmark formats. At the same time, due to simplified delay/power Liberty models and timing constraints, the contests fail to address real-world criteria for gate sizing that are highly challenging in practice. We observe that lack of consideration of practical issues such as electrical and multi-corner constraints – along with limited sets of benchmarks – can *misguide* the development of contest-focused academic sizers. Thus, we study implications of the "gap" between academic sizers and product design use cases. In this paper, we note important constraints of modern industrial designs that are generally not comprehended by academic sizers. We also point out that various optimization techniques used in academic sizers can fail to offer benefits in product design contexts due to differences in the underlying optimization formulation and constraints. To address this gap, we develop a new robust academic sizer, *Sizer*, from a fresh implementation of *Trident* [3]. Experimental results show that Sizer is able to achieve up to 10% leakage power and 4% total power reductions compared to leading commercial tools on designs implemented with foundry technologies, and 7% leakage power reduction on a modern industrial design in the multi-corner multi-mode (MCMM) context.

1. Introduction

Discrete gate sizing, i.e., change of gate width and length, and Vt type, has been widely adopted for timing and power optimization. Gate sizing optimization can be applied at every design stage, e.g., post-synthesis, post-placement and timing ECO. It is especially suitable for late-stage, post-routing optimization since it incurs relatively small perturbation to the design netlist and layout compared to other optimizations such as logic restructuring and buffer insertion. The importance of gate sizing optimization has been emphasized by both industry and academia for a number of years. The 2012/2013 ISPD gate sizing contests [1,2] have given practical impetus to academic research, using industry-standard benchmark data formats and constraint types. The ISPD contest enablement spans consideration of interconnect parasitics, maximum transition time (*MaxTran*) and capacitance (*MaxCap*) constraints, and use of a commercial signoff timer for timing analysis. Several academic sizers achieve good performance on the contest benchmarks. However, due to highly artificial gate delay/power modeling, as well as the lack of real-world timing constraints, winning codes are unlikely to be able to handle core challenges of gate sizing optimization in modern industrial designs. Indeed, it is our observation that the simplified constraints in contest benchmarks can potentially drive academic sizers in wrong directions. This paper describes our experience in identifying and overcoming this mismatch, as we evolved a successful academic sizing approach to perform well in an industry design context.

[*] Corresponding author. ECE Department, University of California at San Diego, La Jolla, California, USA.
E-mail addresses: hamed.fatemi@nxp.com (H. Fatemi), abk@ucsd.edu (A.B. Kahng), hyein.lee1@synopsys.com (H. Lee), jiajial@qti.qualcomm.com (J. Li), jose.pineda.de.gyvez@nxp.com (J. Pineda de Gyvez).

https://doi.org/10.1016/j.vlsi.2019.01.008
Received 29 November 2018; Accepted 25 January 2019
Available online XXX

Limitations of academic sizers. We observe that the academic sizing context has several potential inadequacies or gaps with respect to commercial use cases, including the following. While an academic contest will never match the "real world", we wish to highlight gaps that potentially *mislead* academic research efforts' identification of promising directions.

(1) **Important constraint scenarios are overlooked in the academic context**. Notably, constraints given by academic contests do not include multi-corner multi-mode (MCMM) timing analysis. MCMM timing signoff is essential for modern product designs, where chip designs typically operate (i) under various operating conditions with different temperatures and voltages, (ii) in multiple functional scenarios such as sleep mode and active mode, and (iii) in a regime of manufacturing process variations [4,5]. In digital timing signoff, a *timing corner* represents a particular combination of process, voltage and temperature (PVT) status, and corresponds to a set of timing libraries (Liberty) characterized for that dedicated PVT corner. A *mode* represents a functional scenario, and is characterized by timing constraints (.sdc). A *timing view* is defined by a pair of a timing corner and a mode [4,5]. In the MCMM context, multiple timing views are considered, and must be simultaneously comprehended by gate sizing optimization.

In the MCMM context, the electrical properties of each gate, and of its driven net, vary over different corners. We discuss below how achieving and maintaining timing signoff across multiple corners and modes (e.g., nominal, turbo and test modes) seems to require very different optimization strategies from optimizations that are successful for a single corner/mode. Further, a gate sizing optimization that does not comprehend multiple corners/modes cannot – to our knowledge – achieve a timing-legal solution for all corners and modes. Handling of other constraint types, such as multiple clock/power domains, or timing exceptions, seems to be further removed from the core heuristic design of a gate sizing tool.

(2) **Academic sizers are not "robust" across different designs and technologies.** Due to the nature of contests, academic sizers can be "over-trained" with the specific set of testcases and objectives that are given by a contest [6] evaluates a commercial sizer and a contest-winning academic sizer with both contest benchmarks and real designs synthesized with foundry technologies. The academic sizer is observed to perform better on contest benchmarks while showing worse results on real designs, as compared to the commercial sizer. The authors of [6] point out that the change in tools' relative superiority across technologies raises the possibility that the academic sizer might be specialized to the contest benchmark designs.

In our experience, an academic sizer that applies strategies particularly adapted to contest benchmarks may not be *capable* of producing good solutions for real-world designs. The reasons are (i) real designs can differ from contest designs with respect to timing slack distribution, netlist structure, instance count, etc.; and (ii) academic sizers trained for a particular contest technology may not perform well with different technologies since the electrical characteristics (i.e., leakage/dynamic power-delay tradeoff curves of standard cells) can differ meaningfully across process technologies. We have also found that (iii) the simplified and artificial delay/power Liberty model of academic contests cannot capture meaningful electrical attributes of production cell libraries. More specifically, nonlinear and state-dependent power and delay values, nonlinear input capacitance values, and multiple power/ground pins, which are generally seen in foundry Liberty models, are absent from the ISPD contest Liberty models. Due to unrealistic Liberty models, contest benchmarks might not expose core challenges of sizing optimization in product designs. For instance, fixing MaxTran violations in an MCMM context was not comprehended. This creates a risk of misdirecting considerable academic research effort.

(3) **Sizing contests typically do not require support for standard formats of Liberty and design files**. Examples of standard formats include Verilog netlist (.v), extracted interconnect parasitics (.spef), and timing constraints (.sdc). Typically, contest organizers provide simplified versions of such files with a parser that only comprehends the simplified input formats. (Yet, open-source and arguably more robust parsers for the full standard formats are available, e.g., Refs. [7–9].) In our view, the lack of insistence on support for standard formats unnecessarily hampers transfer of academic sizers to real-world design applications. This necessitates workarounds such as those developed in Ref. [6]. Importantly, efforts that chain together the executables of entries from multiple academic contests are blocked from assessment in real-world contexts.

Our work. In this paper, we present key learnings from a multi-year "journey" to make an academic sizing tool applicable to, and yield benefits for, a real industrial IC. We describe aspects of modern industrial designs that are generally not comprehended by academic sizers, but that strongly affect choice of optimization and metaheuristic techniques. We also observe how various optimization techniques used in academic sizers might not be appropriate for product designs due to practical issues such as runtime. We have addressed such gaps between contest-driven research and the real-world application by developing *Sizer*, a new, robust gate sizing optimization tool that incorporates a near-complete change of techniques as compared to our starting point of Trident [3]. The transition from an academic contest setting to real-world designs shows that techniques beneficial in the contest setting may not have benefit for real designs – forcing the development and tuning of a number of new techniques. Ultimately, Sizer achieves 7% leakage power reduction over the commercial tool's high-effort solution on a production design, with signoff at over two dozen mode-corner combinations.

Our contributions are summarized as follows.

- We highlight gaps between academic sizers and commercial sizers due to missing real-world constraints and the characteristics of industrial designs. We describe challenges of transfer/productization that include MCMM timing signoff, maximum transition time constraints, hold time constraints, and complex timing structure of a product design.
- We suggest that the "competitive landscape" of the gate sizing optimization – including aspects that are particularly challenging to academic approaches – should be better conveyed to the research community. Our experiences with Sizer highlight how obliviousness to aspects of real-world application such as practical runtime/memory limits, input-dependent performance models, etc. can easily prevent assessment of an academic sizer within a commercial design context.
- We develop Sizer, a new academic gate sizing tool that is applicable to modern industrial designs. Results reported below show that Sizer achieves solution quality improvement over high-effort commercial tool results, and achieves benefits for a real industrial IC. Sizer embodies a near-total change of techniques as compared to the starting point of Trident [3], which had been successful at the ISPD-2013 gate sizing contest [2].
- We implement dynamic and total power estimations which enable Sizer to go beyond the original contest optimization objective i.e., leakage power-only optimization, and to smoothly control the tradeoff between dynamic and leakage power optimization.
- We compare Sizer with two leading commercial sizers.[1] We achieve 7% leakage power reduction over the high-effort solution of a commercial tool on a design from *NXP Semiconductors* [13]. We also successfully apply Sizer to various testcases synthesized with different foundry technologies.

- We study the impact of various sensitivity functions on the solution quality of Sizer and provide intuition regarding the choice of sensitivity functions.

The remainder of this paper is organized as follows. In Section 2, we review related work. Section 3 summarizes key practical constraints in industrial designs that we believe strongly affect heuristic design in a sizing tool. We introduce our new techniques in Section 4. Our overall optimization flow is described in Section 5. We present experimental results in Section 6 and conclude in Section 7.

2. Related work

Reflecting the importance of the application, numerous algorithms for both continuous and discrete gate sizing optimizations have been proposed in the literature. Earlier works have focused on continuous gate sizing that optimizes parameters of transistors [40] or of standard cells, such as drive strength, input-pin capacitance, etc. Discrete or library cell-based gate sizing works have applied a wide variety of optimization techniques – Lagrangian relaxation (LR) [17–23,25–29]; dynamic programming (DP) [30,31]; sensitivity function-based optimization (SF) [3,35,36,37–39]; branch and bound (BB) [32]; linear programming (LP) [14–16]; parallel and randomized algorithm (RA) [34]; and Simulated Annealing (SA) [33]. Table 1 taxonomizes previous gate sizing work. Columns 2–8 contain the frameworks that are used for each gate sizing approach. Columns 9–12 show the objective function of each work. Columns 13–16 show the important considerations for real-world gate sizing. The last column shows whether the literature applies its approach to real product designs. An overview of selected recent literature for two popular frameworks, LR and SF, is as follows. Some previous works cannot be categorized into a single particular framework, since more than one frameworks or techniques are used. We attempt to categorize each reference according to the predominant framework or technique used.

LR-based approaches. A number of recent works use LR-based methods for gate sizing optimization. A successful recent application of LR-based gate sizing optimization for industrial designs is described by Ozdal et al. [23,24]. In Refs. [23,24], a cost function comprehending the tradeoff between power and timing slack is formulated and then relaxed to a Lagrangian subproblem by Karush-Kuhn-Tucker conditions as in Ref. [17]. The subproblem is modeled as a graph problem and solved by using critical tree extraction and DP-based optimization. The work of [23,24] is notable for its thorough treatment of real-world issues such as those we highlight. But, details of implementation are not available to the research community. Subsequent academic work driven by Refs. [1,2] does not capture real-world issues as [23,24] do. Huang et al. [19] suggest a method to obtain Lagrangian multipliers based on the timing history of previous iterations to improve the conventional subgradient-based method. Rahman et al. [25] use an extended logical effort for gate delay modeling to formulate LR problems to which dynamic programming is applied. Flach et al. [20] propose a gate sizing optimization flow that combines LR-based framework and various heuristics. In Ref. [20], the LR problem is solved by a sensitivity-based approach; greedy timing and power optimization is subsequently performed. Reimann et al. [26] extend [20] to adapt the LR-based gate sizing to industrial designs. In Ref. [26], runtime scalability, preserving timing quality and incremental optimization are considered. Roy et al. [27] solve the gate sizing problem in the context of multiple operating conditions. They extend the LR-based gate sizing approach of [17,23] to support multiple scenarios. More specifically, the authors of [27] use the weighted sum of leakage and dynamic power across different operating conditions as the objective function.

We note that many of LR-based works construct simple *analytic* gate delay models to encompass the discrete gate sizes found in Liberty gate timing libraries. Such analytic gate delay models might not be accurate due to nonlinear characteristics of gate delays. The suboptimality

Table 1
Summary of works on gate sizing optimization.

Work	Year	Framework							Objective				Interconnectdelay	MaxTran	MaxCap	MCMM	Realdesigns
		LP	LR	DP	SF	BB	SA	RA	Leakage	TotalPwr	Delay	Area					
[14]	90	✓									✓						
[15]	05	✓								✓			✓				
[16]	09	✓							✓								
[17]	99		✓									✓	✓				
[18]	05		✓						✓								
[19]	11		✓							✓			✓				
[20–22]	14, 12, 14		✓						✓				✓	✓	✓		
[23,24]	11, 12		✓									✓	✓	✓	✓		✓
[25,26]	13, 15		✓							✓			✓				✓
[27]	15		✓						✓	✓			✓			✓	
[28]	02		✓								✓		✓				
[29]	15		✓						✓				✓				
[30]	09			✓								✓					
[31]	10			✓								✓	✓				
[32]	11					✓				✓			✓				✓
[33]	13						✓		✓	✓					✓		
[34]	09							✓	✓				✓				
[35]	06				✓				✓				✓				
[36,37]	12, 12				✓				✓				✓	✓	✓		✓
[3]	13				✓				✓			✓	✓	✓	✓		✓
[38,39]	04, 00				✓					✓							
our work					✓				✓	✓			✓	✓	✓	✓	✓

caused by the combination of inaccurate delay models and intrinsic discreteness of the gate sizing problem is one of the major limitations of LR-based approaches. Furthermore, for industrial designs, model-based mathematical methods may be inefficient due to complex constraints such as MaxTran and MaxCap.

SF-based approaches. Wei et al. [39] use the sensitivity function (SF) approach (i.e., iteratively making discrete sizing moves to follow the gradient of a given SF) for simultaneous sizing and dual-Vt assignment. Srivastava et al. [38] propose a dual-Vdd and dual-Vt assignment method based on sensitivity calculations. Gupta et al. [35] use the sensitivities of leakage and delay to gate-length biasing for leakage power optimization. Rahman et al. [37] apply a cost function that considers total slack and leakage changes. Kahng et al. [36] propose sensitivity-guided metaheuristics based on sequential importance sampling and a multistart technique with various sensitivity functions to optimize gate sizing and Vt flavor for minimized leakage. In follow-on work, the optimizer is improved with an efficient and accurate internal timer based on various delay models [3]. To ensure the accuracy of the internal timer, timing information is correlated to the signoff timer's analysis *during* the gate sizing optimization, using techniques proposed in Refs. [41,42].

Our work is distinguished from previous literature in that – to the best of our knowledge – it is the first academic work, available to public [43], that simultaneously addresses all the essential constraints for industrial designs. Further, we provide evaluation using a product design in an industry context.

3. Constraints for modern industrial designs

We now review examples of practical constraints that are not emphasized in previous published works and academic contests, but are critical for modern industrial designs.

Ozdal et al. [23,24] give a comprehensive summary of the main optimization challenges for gate sizing in modern industrial designs. These challenges are formulated in the ISPD-2012 and the ISPD-2013 Gate Sizing Contests [1,2]; many practical considerations such as use of a golden signoff timer, interconnect parasitics, maximum transition and capacitance constraints are addressed in the provided testcases. However, in addition to unrealistic Liberty, we find that there are several missing constraints that must be considered for a product design in industry.

Multi-Corner Multi-Mode (MCMM). For modern product designs, a number of PVT corners, along with various functional modes, must be considered during timing signoff. Due to the different delay and power characteristics of cells across multiple corners, it is difficult to achieve a converged solution in the MCMM context. In other words, a signed-off netlist at a particular corner and mode could have timing violations at other corners and modes. Furthermore, there exist "ping-pong" situations where an upsizing move for setup timing recovery in a timing view (a pair of PVT corner and mode) causes timing violations in another timing view. Fig. 1 shows an Example of the ping-pong situation. We assume that C_2 and C_4 are on the setup timing-critical path in *view*1, and that C_1, C_3, C_5 and C_6 are on the timing-critical path in *view*2. To reduce the critical path delay in *view*1, C_2 must be upsized. However, the increased input capacitance of C_2 increases the load capacitance of C_1 (a fanin cell of C_2) and thus increases C_1's delay. As a result, a timing violation occurs in *view*2. Similarly, upsizing cells in *view*2 can create timing violations in *view*1. In modern process technologies, the ping-pong situations become significantly worse with the explosion of PVT corners [44]. A gate sizing optimization must comprehend the timing impact of each Vt swapping or width sizing move in all views simultaneously, and be able to avoid the ping-pong situations efficiently to obtain a converged solution.

Importance of transition time. The maximum transition (MaxTran) constraint is an upper limit for the transition time at a pin of a gate [1,2,4,5]. The MaxTran constraints are specified in timing libraries or via the set_max_transition command in EDA tools [4,5]. The actual transition time is checked against the MaxTran constraint for every pin in the netlist. For Example, if the input pin of an INVX1 instance has a MaxTran constraint of 10, and the actual transition time at the input pin is 15, then the pin violates the MaxTran constraint.

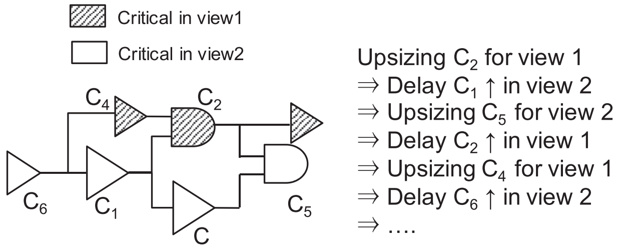

Fig. 1. Example showing a ping-pong situation during timing recovery for MCMM.

Fixing MaxTran violations is important for product designs for (i) an accurate timing analysis, and (ii) total power optimization. More specifically, in NLDM (Non-Linear Delay Model)-library-based timing analysis, which is still widely used in mature process technologies, violating the MaxTran constraint can induce an inaccurate cell delay calculation due to extrapolation of the cell delay table. A large transition time also increases the internal power of gate instances and harms netlist quality in terms of total power.

The ISPD contest benchmarks [1,2] include MaxTran constraints as one of the criteria for legal solutions. However, due to the unrealistic timing library, MaxTran violations are relatively easy to fix in the contest technology. For Example, the sensitivity of the output transition time of a cell to its load capacitance is relatively small compared to the sensitivity to its drive strength. Thus, in the contest technology, upsizing a cell will always trivially cure a MaxTran violation without causing any new violation on its upstream driving cell. However, in reality, depending on netlist topology (e.g., having many high-fanout nets) and technology, MaxTran constraints can be easily violated and the violations are difficult to fix. For example, upsizing cells causes transition time violations at the output pins of their fanin cells due to load capacitance increase. Thus, a viable strategy for fixing MaxTran violations must comprehend both the topology of the input netlist and the ripple effect of upsizing cells. We observe that in practice, the MaxTran constraint becomes a "first-class" concern at 65 nm and below enablements; a gate sizing optimization that is oblivious to such constraints can result in numerous violations, even when other timing constraints such as setup and hold are met.

Hold time consideration. Hold time violations are more critical than setup time violations in product designs since the functionality of a design fails if any hold time violation exists. Despite its importance, the hold time constraint has not been emphasized in academic benchmarks due to the nature of gate sizing for power optimization; most cells are downsized and replaced with higher-Vt cells, which are less likely to incur many hold time violations. However, hold time violations become more critical in the MCMM context, as well as during timing recovery and MaxTran fix optimizations. In particular, upsizing width or decreasing Vt during the timing recovery can lead to hold time violations in a hold-critical timing view, depending on the structure of timing paths in the input design.

Complex structure of a product design. A product design is more complicated than academic benchmarks in many aspects, as it may involve multiple power domains, multiple clocks, and the existence of memories and macros. To support multiple power domains, instances in different power domains must be analyzed with multiple timing/power Liberty tables. Different clock periods due to multiple clocks per timing endpoint such as flip-flops and primary outputs must be handled for an accurate timing analysis. Additional handling in timing analysis is required for memories and macro blocks. That is, the timing graph of a

Table 2
Comparisons of Trident [3] (Tri) and this work (Szr).

Techniques	Tri [3]	Szr
MCMM-aware static timing analysis	–	✓
MCMM-aware timing recovery	–	✓
MCMM-aware sensitivity function	–	✓
Transition time correlation	–	✓
Total power estimation	–	✓
Dynamic change of sensitivity function	–	✓
Prioritization of moves	–	✓
Kick move	✓	✓
Go-with-the-winner (GWTW)	✓	–
Peephole optimization	✓	–

design must properly capture.sdc constraints pertaining to timing paths involved with memories or macros. Lack of understanding of memories and macro blocks in an academic sizer results in significantly degraded solution quality with inaccurate timing slack values.

4. Our techniques

We now describe key techniques incorporated in Sizer. Driven by practical constraints seen in product designs, we introduce new features and techniques including (i) MCMM-aware SF; (ii) total power estimation; and (iii) MaxTran violation fixing.

4.1. Comparisons to Trident [3]

Trident is an academic sizer originally developed for the ISPD-2013 Gate Sizing Contest [2]. The core engine is based on sensitivity-guided metaheuristics. A multistart technique (*go-with-the-winners*, or GWTW [45]), kick moves (following the *large-step Markov chain* optimization technique [46]) and *peephole optimization* are used in Trident. The GWTW metaheuristic performs repeated optimizations within randomized multistarts to effectively explore a large search space. More specifically, Trident runs multiple optimizations concurrently, guided by different sensitivity functions, and records the best solution at prescribed intervals. The kick move technique, which originated in large-step Markov chain optimization, attempts to make a large change, i.e., by upsizing many cells at once, to escape local optima and reach a solution that is closer to a global optimum. The peephole optimization entails exhaustive search for the best size combination for a small path-connected series of cells.

Table 2 summarizes the optimization techniques used in Trident ([3]) and Sizer (our work). We do not include the sensitivity function (SF) framework itself in the table since SF-based gradient-following (greedy, steepest descent) algorithm is not new to Ref. [3]; it was in the SensOpt package [47] from which [3] was derived. The Blaze MO tool [12] is another Example of sensitivity-based sizer. MCMM-awareness is our key improvement to support product designs. Data structures in Sizer store timing information for each timing view. New sensitivity functions for MCMM are developed to enable effective MCMM-aware optimization for both timing recovery and power reduction. Total power estimation is added as well, to guide the total power optimization. Additionally, with the increased importance of transition time in real designs, we notice that obtaining accurate transition time is essential for the optimization. We thus implement the internal timer of Sizer to improve the accuracy of transition time. We adopt the transition time correlation method of [42], whereas Trident [3] performs only slack correlation.

We note that a number of distinguishing innovations in Ref. [3] such as the use of GWTW and peephole optimization were eventually discarded in Sizer because no benefit could be seen in the production context. For realistic designs with many (i.e., >30) timing views and multiple clocks, the optimization complexity increases dramatically. For Example, the GWTW paradigm requires additional cores according to the number of multistarts, and each individual optimization requires multiple timers to support MCMM. Thus, GWTW demands huge computing resources in the production context, due to its greater complexity of timing analysis. Peephole optimization is also computationally demanding, since it attempts to evaluate multiple combinations of solutions for several cells at a time. Such approach is infeasible if computing resources and runtime are limited. Crucially, our background experiments showed that even with huge computing resource consumption, the two techniques have little or no benefit in terms of solution quality. For example, the additional leakage reduction obtained from GWTW is 1% at the cost of 2.5 × runtime, for the aes design. For the m0 design, 3% more leakage reduction can be achieved, but at the cost of 4.3 × runtime. With use of peephole optimization, our background studies found that the final leakage power worsens for both the *aes* and *m0* designs, by 2.6% and 1.0%, respectively.

Table 3
Notations.

Notation	Meaning
P	weighted sum of leakage and dynamic power
D	clock period
TNS	total negative setup time slack
WNS	worst negative setup time slack
s_i	setup time slack of cell i
d_i	delay of cell i
c_i	load of cell i
$tran_j$	transition time at pin j
$tran_{\max}$	maximum transition time
s_i^{tran}	maximum transition time slack of cell i
$\#paths_i$	number of register-to-register paths going through cell i
β_v	weighting factor for view v for MSF3
γ	normalizing factor for transition time slack

4.2. New sensitivity functions in the MCMM context

Sensitivity functions (SFs) are used as guidance to select sizing *moves* (Vt swapping or sizing) that give the maximum power reduction benefit with the minimum timing impact. [2] For this purpose, an SF should calibrate the power benefit at the expense of timing slack degradation, or vice versa. However, it is not straightforward to estimate timing or power impact of a sizing move on the design since (i) the entire timing graph, which is very complex in modern product designs must be comprehended due to the nature of timing calculation propagation toward downstream cells, and (ii) in the MCMM context, variations across multiple corners must be considered as discussed in Section 3.

To address these challenges, we introduce new estimations of timing and power impact, considering MCMM, for our SFs as follows.

Table 3 summarizes the notations that we use in our SFs. Each notation can be extended by adding subscript v to represent the notation associated with timing view v. I.e., $s_{i,v}$ is slack of cell i in timing view v.

- P denotes a weighted sum of leakage and dynamic power. We provide a detailed discussion of power calculation in Section 4.3 below.
- s_i denotes the timing slack of cell i. We consider only setup time slack for our SFs. We consider hold time differently in our optimizer. We revert any Vt-swapping/sizing move that results in any hold time violation. This is for two main reasons. First, during the power reduction stage, due to the nature of power optimization where cells are downsized and/or swapped to higher Vt cells, delays of the optimized cells typically increase, which is actually better for timing paths with respect to hold time. Second, during timing recovery, as we seek to cure setup-violating paths, the target cells for upsizing are mostly setup-critical. [3] Also, inclusion of hold time increases computation complexity.

Table 4
Sensitivity functions for a single view and MCMM.

Type	Index	Equation
Single	SF1	ΔP
	SF2	$\Delta P \cdot s_i$
	SF3	$\Delta P/\Delta d_i$
	SF4	$(\Delta P \cdot s_i)/\#paths_i$
	SF5	$(\Delta P \cdot s_i)/(\Delta d_i \cdot \#paths_i)$
	SF6	$-\Delta P/(\Delta s_i \cdot \#paths_i)$
MCMM	MSF1	$\Delta P \cdot \min(\min_{v \in views}(s_{i,v} - \Delta d_{i,v}), \gamma \cdot \min_{v \in views} s_{i,v}^{tran})$
	MSF2	$(\Delta P \cdot \min(\min_{v \in views}(s_{i,v} - \Delta d_{i,v})/c_{i,v}, \gamma \cdot \min_{v \in views} s_{i,v}^{tran}/c_{i,v}))$
	MSF3	$\Delta P \cdot \#paths_i \cdot sum_{v \in views} \boldsymbol{\beta}_v/\Delta d_{i,v}$

Table 5
Parameter conditions and their implications.

Index	Parameter Condition				Implication
	ΔP	s_i	Δd_i	Δs_i	
1	±	±	+	+	N/A
2	±	±	–	–	
3	–	±	–	+	Always pick
4	+	±	+	–	Always avoid
5	+	–	–	+	Timing recovery (TR)
6	+	+	–	+	Avoid (Aggressive TR)
7	–	+	+	–	Power reduction (PR)
8	–	–	+	–	Avoid (Aggressive PR)

- d_i denotes the delay of cell *i*. For Δd_i in SFs, we calculate the delay of each input to output timing arc of cell *i* before and after a given move, and return the largest arc delay change.
- c_i is the output load of cell *i*; this output load includes wire capacitance and the sum of input pin capacitances of cell *i*'s fanout cells.
- s_i^{tran} is the minimum value of $(tran_{\max} - tran_j)$, over all pins *j* of cell *i*.
- $\#paths_i$ is the number of register-to-register paths that pass through cell *i*. $\#paths_i$ is calculated by multiplying the number of downstream registers and the number of upstream registers of cell *i*.

Table 4 shows SFs for single-view and MCMM optimizations. ΔP, Δs_i and Δd_i denote the differences in *P*, s_i and d_i values, respectively, before and after a move of cell *i*. For Example, ΔP is the power (total or leakage power) of the design after a move subtracted by the original power value.

SF for a single view. In Table 4, SF1 simply prioritizes cells that offer large power reduction. SF2 prioritizes cells with large timing slacks that also offer large power reduction. SF3 selects cells that have a smaller delay penalty but a larger power reduction. SF4 is a variation of SF2, but we add *#paths* so that we do not pick cells that affect slacks of many timing paths. SF5 is a combination of SF3 and SF4. SF6 picks cells that have less impact on total negative slack (TNS) but have more power reduction. To reduce runtime, we estimate ΔTNS as $(-\Delta s_i \cdot \#paths)$.

SF for MCMM. Simultaneous consideration of multiple timing views is important since (i) power-critical views are different from timing-critical views, and (ii) timing must be signed off in all the given timing views. Thus, SFs always need to consider the "worst" timing/power impact considering each of the given timing views. We propose new SFs to guide the optimizations in the MCMM context, i.e., MSF1, MSF2 and MSF3, as shown in Table 4. Here, ΔP in all MSFs is calculated in the most power-critical view. MSF1 prioritizes cell moves that lead to large setup time slack $(s_i - \Delta d_i)$ and MaxTran slack, in conjunction with large power reduction. For the setup time slack and MaxTran slack, the minimum values across all timing views are considered. For γ, normalizing factor for MaxTran slack, we empirically use 0.5 in our reported studies based on results of background experiments. MSF2 is a variation of MSF1. In MSF2, c_i is used as a denominator to avoid downsizing cells with larger load capacitance (or many fanout cells). This helps to avoid MaxTran violations.

MSF3 uses weighting factor $\boldsymbol{\beta}_v$ to prioritize timing-critical views and thus address different timing constraints and slacks for each timing view. We study three weighting factors: (i) $\frac{1}{D_v}$; (ii) $\frac{TNS_v}{D_v}$; and (iii) $\frac{WNS_v}{D_v}$. Based on our experimental results on design *m0* in 28 nm LP technology, using weighting factors (i), (ii) and (iii) respectively leads to 26%, 13% and 4% less runtime needed to achieve a timing-feasible solution, compared to timing recovery without weighting factors.

Example. The following example illustrates how SF values for a single timing view and MCMM are calculated. Let us assume three timing views, i.e., v_1, v_2, v_3, and v_3 is the power-critical view. The corresponding power, setup time slack, delay, transition time slack, *#paths* and output load for cell *i* are summarized as follows.

- Setup time slack: $s_{i,1} = 10$, $s_{i,2} = 5$, $s_{i,3} = 20$
- Transition time slack: $s_{i,1}^{tran} = 20$, $s_{i,2}^{tran} = 16$, $s_{i,3}^{tran} = 40$
- ΔPower: $\Delta P_1 = 3$, $\Delta P_2 = 4$, $\Delta P_3 = 6$
- ΔDelay: $\Delta d_1 = 6$, $\Delta d_2 = 3$, $\Delta d_3 = 1$
- The number of paths: $\#paths = 5$
- Output load: $c_{i,1} = 2$, $c_{i,2} = 2$, $c_{i,3} = 2$
- $\boldsymbol{\beta}$: $\boldsymbol{\beta}_1 = 0.3$, $\boldsymbol{\beta}_2 = 0.2$, $\boldsymbol{\beta}_3 = 0.1$

Each SF considering view v_1 only is calculated as follows.

- SF1: $\Delta P_1 = \Delta P = 3$
- SF2: $3 \cdot 20 = 60$
- SF3: $3/6 = 0.5$
- SF4: $3 \cdot 20/5 = 12$
- SF5: $3 \cdot 20/(6 \cdot 5) = 2$
- SF6: $-3/(20 \cdot 5) = -0.03$

And, each MSF considering all timing views, i.e., v_1, v_2 and v_3, is calculated as follows.

- MSF1: $\Delta P = \Delta P_3 = 6; \min_v s_{i,v} - \Delta d_{i,v} = 2; \min_v s_{i,v}^{tran} = 16; MSF1 = 6 \cdot \min(2,8) = 12$
- MSF2: $\Delta P = \max_v \Delta P_v = 6; \min_v(s_{i,v} - \Delta d_{i,v})/c_{i,v} = 1; \min_v s_{i,v}^{tran}/c_{i,v} = 8; MSF2 = 6 \cdot \min(1,4) = 6$
- MSF3: $6/5 \cdot (0.3/6 + 0.2/3 + 0.1/1) = 0.26$

The complexity of SF and MSF computation for a single cell are O{1} and O(N), respectively, where N is the number of timing views.

Parameter conditions for SF usage. Table 5 summarizes parameter conditions and their implications. "+" means a positive value, "-" means a negative value. Δd_i and Δs_i cannot be both positive or negative, so we do not consider such cases (Conditions 1 and 2). If fanin and/or fanout cells of target cell *i* are affected, Δd_i and Δs_i can be both positive or both negative. However, such cases are extremely rare in our experiments. We always pick moves that provide both power and timing benefits (Condition 3). Similarly, we avoid moves leading to both power and delay penalties (Condition 4). For timing recovery, we use SF6 and MSF3 if Condition 5 is met. We do not consider cells in Condition 6 for upsizing or decreasing *V t*. During power reduction, we calculate SFs for a move such that Condition 7 is met. SF1-6, MSF1-2 are used for such cases in our experiments. For Condition 8, we do not allow downsizing/increasing *V t* since aggressive power reduction leads to excessive timing violations.

4.3. *Total power estimation*

In this subsection, we describe Sizer's capability to perform total (weighted) power reduction. To comprehend the total power optimization, we consider both leakage and dynamic power. Dynamic power consists of net switching power and cell internal power.

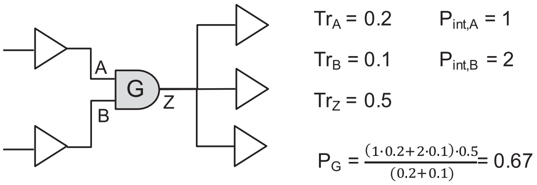

Fig. 2. Example internal power estimation of a 2-input cell.

For leakage power calculation, we directly use the leakage value of each cell from Liberty files. Since leakage is state-dependent in the Liberty files, we use the average leakage value across different states as an approximation. We empirically observe that such approximation overall improves optimization runtime while offering relatively accurate leakage estimation. If state probabilities are available from dynamic simulation and power analysis, then replacing the arithmetic mean across states with a weighted average would be straightforward.

We estimate the net switching power of a net as $\frac{CV^2 \mathrm{Tr}}{2}$, where C is the total net capacitance, i.e., the sum of wire capacitance and the capacitance of cell input pins connected to the output of the wire, V is the supply voltage, and Tr is the toggle rate of the net. We extract the toggle rate and wire capacitance of each net from a golden signoff timer; input pin capacitances of cells are obtained from Liberty files.

For cell internal power, we obtain the values from lookup tables (LUTs) in Liberty files based on input slew and output load of the cell. We note that there exist multiple internal power LUTs for a given cell, where each table corresponds to a particular transition arc, that is, a pair of an input pin and an output pin. As our power analysis is vectorless, we normalize the internal power values related to different input pins based on the toggle rates at pins of the cell. An Example of a 2-input cell is given in Fig. 2, where cell G has input pins A and B. We estimate the internal power of cell G (P_G) as follows.

$$P_G = \frac{(P_{\mathrm{int},A} \cdot Tr_A + P_{\mathrm{int},B} \cdot Tr_B) \cdot Tr_Z}{Tr_A + Tr_B} \tag{1}$$

Here, (i) $P_{\mathrm{int},A}$ and $P_{\mathrm{int},B}$ are respectively the internal power values related to input pins A and B based on LUTs, and (ii) Tr_A, Tr_B and Tr_Z are respectively the toggle rates at input pins A, B and output pin Z. In the Example, P_G is calculated as 0.67 according to Eq. (1).

Note that for every sizing or Vt swapping, pin capacitance and slew values change and thus the net switching power and internal power change. To adapt our sensitivity functions for total power optimization, we replace the power terms P in SFs in Section 4.2 with $\alpha \cdot P_{leak} + (1 - \alpha) \cdot (P_{\mathrm{int}} + P_{sw})$, where P_{leak}, P_{int} and P_{sw} indicate leakage power, internal power and net switching power respectively. α is a weighting factor to trade off between leakage power and dynamic power optimization. A larger (resp. smaller) α value leads to a greater reduction in leakage power (resp. dynamic power). To our knowledge, previous literature on gate sizing optimization for total power optimization typically uses gate capacitance or transistor width for dynamic power estimation. By contrast, to achieve a more accurate total power estimation, we follow industrial standard practice and estimate dynamic power based on both net switching power and cell internal power.

4.4. Handling maximum transition constraints

Our Sizer has two specific features to handle MaxTran violations: (i) transition time correlation, and (ii) a dedicated procedure for fixing MaxTran violations.

As pointed out in Section 4.1, Trident [3] performs slack correlation between its internal timer and the signoff timer for a more accurate slack estimation. However, lack of transition time correlation in its optimization can result in additional MaxTran violations due to transition time mismatches. We thus perform transition time correlation in Sizer using the method of [42], in addition to the slack correlation.

Algorithm 1 Procedures for fixing MaxTran violations.

```
Procedure: FixMaxTran(netlist, max_tran)
Input: netlist netlist, MaxTran constraint max_tran
Output: optimized netlist with reduced transition time violations
1:  for all cell $c \in$ netlist, in topological order do
2:    for all violating pin $vp \in c$ do
3:      if $vp$ is an output pin of $c$ then
4:        for all focell $\in$ fanout cells of $c$ do
5:          downsize focell
6:          if $s_c < 0$ then revert the move
7:          if $tran_{vp} \leq$ max_tran then break
8:        end for
9:        while $tran_{vp} >$ max_tran do
10:         UpsizeCellforTran(vp, c)
11:       end While
12:     else
13:       ficell ← the fanin cell of the input net of vp
14:       while $tran_{vp} >$ max_tran do
15:         UpsizeCellforTran(vp, ficell)
16:       end While
17:     end if
18:   end for
19: end for
    Procedure: UpsizeCellforTran(vp, cell)
    Input: violating pin vp, cell cell
    Output: updated netlist
20: SF_size ← $\Delta tran_{vp} / \Delta P$ (when upsized)
21: SF_vt ← $\Delta tran_{vp} / \Delta P$ (when Vt swapped)
22: if SF_size = = 0 and SF_vt = = 0 then return
23: if SF_size < SF_vt then
24:   upsize cell
25: else
26:   decrease $Vt$ of cell
27: end if
28: if $s_{cell} < 0$ then revert the move
```

Algorithm 1 shows procedures to fix MaxTran violations. In the *FixMaxTran* procedure, if the violation occurs at an output pin, we visit all fanout cells of the violating pin (*focell*), and perform downsizing on each of the fanout cells until the maximum transition violation is fixed, unless the downsizing move leads to a timing violation (Lines 4–8). If downsizing moves of the fanout cells do not fix the maximum transition violation, we perform upsizing or decreasing Vt of the violating cell by invoking the *UpsizeCellforTran* procedure (Lines 9–11). If the violation occurs at an input pin, we perform upsizing width or decreasing Vt of the fanin cell by invoking the *UpsizeCellforTran* procedure (Lines 14–16). In *UpsizeCellforTran*, we first evaluate the resultant transition time reductions from upsizing and decreasing Vt as well as the corresponding power penalties (Lines 20–21). If the transition time does not improve with both moves, we do not perform upsizing (Line 22). Between upsizing and decreasing Vt, we pick the move that has a larger ratio of transition time reduction to power penalty (Lines 23–25). We revert the move if it leads to a timing violation (Line 28).

4.5. Other techniques for better optimization

In this subsection, we describe several other techniques used to achieve improved optimization.

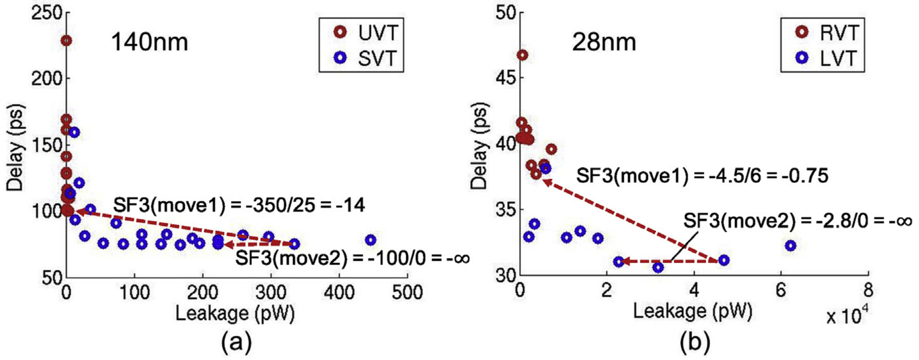

Fig. 3. Leakage vs. delay of Vt swapping and sizing in (a) 140 nm (TT, 1.5 V, 25 °C) (b) 28 nm LP (TT, 1.0 V, 25 °C).

Prioritization of move type. We introduce prioritization of a certain move type on top of the SF-based prioritization for the following two reasons. First, the types of SFs we use cannot differentiate between (i) a move that leads to large power reduction as well as large delay increase, versus (ii) a move that has small impact on both power and delay. Second, at early optimization stages when there is sufficient timing slack for power optimization, (i) is preferred to (ii), in order to achieve faster convergence to the local minima.

Fig. 3 shows the leakage-delay tradeoff of buffers (with two Vt flavors and various gate sizes) in (a) 140 nm and (b) 28 nm LP technologies. The leakage and delay values of higher (resp. lower) Vt cells are shown in red (resp. blue) dots. The delay values of each gate are calculated based on NLDM libraries with a fixed input transition 50 ps and a load equal to four times the input pin capacitance of the driving gate. Fig. 3 shows that the leakage values of higher Vt cells (i.e., UVT and RVT) are lower than those of lower Vt cells (i.e., SVT and LVT). The naming conventions for high and low Vts vary according to particular foundries and technology nodes. In the 140 nm dual-Vt library, UVT (Ultra-high threshold voltage) and SVT (Standard threshold voltage) respectively correspond to high and low Vts. In the 28 nm dual-Vt library, RVT (Regular threshold voltage) and LVT (Low threshold voltage) respectively correspond to high and low Vts.

Vt flavor has much larger impact on leakage power than does gate size. We therefore expect that Vt swapping is a stronger lever than sizing for leakage optimization. However, if we use SF3, move2 in Fig. 3 will be selected over move1, even though move2 is a stronger solution for leakage optimization. For dynamic power optimization, due to its direct impact on total design capacitance, sizing is preferred over Vt swapping. Thus, in the contest of different optimization objectives, we prioritize either Vt swapping or sizing during our optimization. More specifically, we pick the moves with preferred move type among the top-k candidate moves (sorted by the sensitivity scores[4]) to execute. Put another way: we ignore the non-preferred move type during our optimization unless there is no move of the preferred move type among the top-k candidate moves. Note that the value of k trades off the effect of sensitivity function versus that of the preferred move type for selection of optimization moves (e.g., a small k indicates that the optimization honors sensitivity scores more). Based on our separate study, we empirically use $k = 40$ (PR_{max} in Table 6) in our experiments.

Table 6
User-defined input parameters.

Notation	Meaning (default)
SF_p, SF'_p	sensitivity functions for power reduction (MSF1, MSF2)
SF_t	sensitivity function for timing recovery (MSF3)
X	#trials for cell Vt swapping or sizing (20% of total #cells)
α	controls leakage and dynamic power optimizations (0.0)
O_{max}	number of optimization loops {8}
T_{max}	maximum number of iterations of timing recovery {30}
PR_{max}	maximum number of candidate moves for prioritization {40}
S_{max}	threshold for dynamic change of SF {5}
X_k	maximum #trials allowed for kick move (0.05% of total #cells)
th_p	minimum allowed slack threshold for power reduction (−20 ps)
th_k	maximum allowed slack threshold for kick move (20 ps)
δ	threshold for correlation (5% of total #cells)

Dynamic change of SF. In a SF-based optimization, the definition of SF can significantly affect the optimization solution quality. An optimal SF choice is dependent on the status of netlists as well as input design information that is initially given (e.g., foundry technologies or libraries, initial netlists, timing constraints, and optimization objectives). Thus, using a single SF throughout the entire optimization procedure increases the likelihood of getting stuck at local minima. To avoid such a situation, we dynamically change the SF during our optimization to adapt to the status of the input netlist. More specifically, Sizer changes SF whenever the optimization with the current SF cannot achieve further power reduction. Based on our experimental results in foundry 140 nm technology, such a dynamic change of SF achieves ~2% further leakage power reduction.

4.6. Unsuccessful techniques

In this subsection, we describe several techniques that have not shown noticeable benefits in our experiments and were eventually dropped.

Tabu search. We have attempted a form of Tabu search [49] during our gate sizing optimization to avoid being stuck at a local minimum. More specifically, we record three to five most recent sizing/Vt-swapping moves performed in the power reduction (resp. timing recovery) stage and forbid these moves for the following timing recovery (resp. power reduction) stage. We initially thought that this would

[1] The two commercial sizers are from two commercial entities and are selected from among various commercial sizers (standalone, or integrated in a timing or a place-and-route tool) that are listed under Timing Analysis, ASIC Layout and/or Power Analysis and Optimization by the industry analyst firm Gary Smith EDA [10] over the past three years. The universe for this selection includes Synopsys PrimeTime [5], Cadence Encounter Digital System/Innovus [11], Blaze MO [12], and Cadence Tempus [4]. We are unable to identify the tools more specifically due to license restrictions and sensitivity of EDA vendors.

[2] Throughout this work, a "move" indicates a Vt-swapping or sizing move of a cell, not a physical movement in placement.

[3] Even in industrial timing ECO flows, the setup-hold critical paths are typically fixed by engineers manually instead of commercial sizers [48].

[4] We use "sensitivity score" to refer to the values calculated by SF.

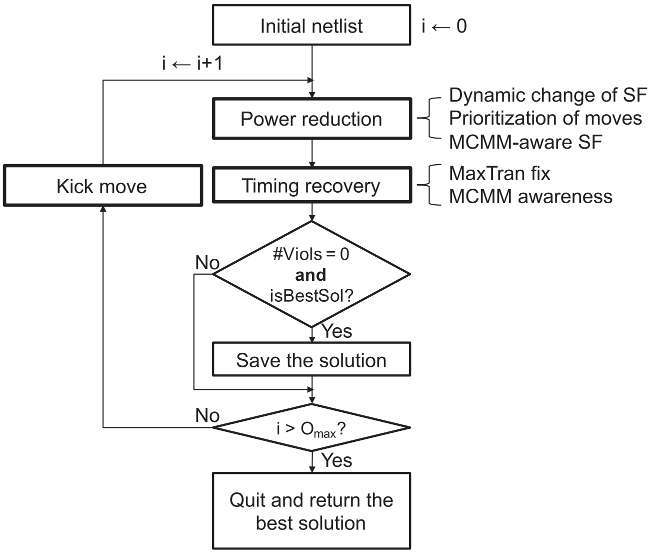

Fig. 4. Overview of the overall optimization flow.

reduce the likelihood of being stuck at a local minimum. However, the Tabu-search-inspired strategy has not helped to improve the solutions for our testcases since there were not many cells that went back to their previous status. We believe that the use of different SFs in the power reduction and timing recovery stages might help avoid previously-visited solutions for each cell.

Smart guardband. We have attempted “smart”, or variable guardband-based, optimization. Here, we refer to a timing margin that is added to timing slack as a guardband. Using a positive (resp. negative) guardband means that timing analysis becomes optimistic (resp. pessimistic) by the amount of the guardband. We have tried varying these guardbands depending on the iteration count within the overall optimization. More specifically, we have used a positive guardband at the initial several iterations of the optimization procedure. We then gradually decrease the guardband at the later iterations of the optimization procedure; in the end, zero guardband is applied so that no timing violation is allowed. In our experiments, this approach has not improved the solution but has only rather increased runtime.[5] Due to the optimism introduced at the first few iterations, Sizer downsizes and/or Vt-swaps gates too aggressively and ends up spending a lot of time on timing recovery.

Multi-step sizing and Vt swapping. We have attempted multi-step sizing and Vt swapping during timing recovery stages. For Example, if 1 × , 2 × and 4 × sizing options are given for a specific gate instance, we have considered 1 × to 4 × (two-step) sizing in addition to 1 × to 2 × (one-step) sizing. Similarly, if LVT, RVT and HVT Vt flavors are given, we have considered LVT to HVT swapping. In our studies, we calculated sensitivity scores for multi-step moves and added them to the candidate list. However, multi-step moves were rarely picked based on the sensitivity scores. Rather, calculating sensitivity scores of these additional moves increases runtime significantly. We conclude that one-step moves are sufficient to consider for timing recovery in our experience.

5. Overall optimization flow

In this section, we describe the overall optimization flow of Sizer and the details of key procedures. Fig. 4 shows the overall optimization flow of Sizer. The optimization flow is an adapted version of [3] with our new techniques that are essential for commercial product designs. The optimization consists of three stages: (i) the sensitivity-based power reduction stage, (ii) the timing recovery stage, and (iii) the kick move stage. In the power reduction stage, Sizer attempts to downsize cells or swap cells to high-Vt cells based on sensitivity scores until timing becomes infeasible in any timing view. In this stage, the dynamic change of SF, the MCMM-aware sensitivity function and the prioritization of moves are applied as discussed in Sections 4.2 and 4.5. In the timing recovery stage, upsizing and/or Vt swapping based on SF is performed to fix any timing violation occurred during the power optimization stage. In the kick move stage, Sizer attempts to upsize/swap cells in order to escape local optima and increase timing slack so that more cells can be downsized. We iterate the optimization loop multiple times and store the best solution (i.e., minimum power solution with legal timing) obtained during the optimizations.

Algorithm 2 The overall optimization flow.

```
Procedure: PowerOpt(netlist, ...)
Input: netlist netlist, user-defined inputs (See Table 6)
Output: an optimized netlist
1:  $best\_sol \leftarrow input$
2:  $SF \leftarrow SF_p$
3:  for $i = 1$ to $O_{max}$ do
4:      $stuck\_count \leftarrow 0$
5:      $PowerReduction(SF, X, th_p)$
6:      for $j = 1$ to $T_{max}$ do
7:          if no timing violation exist then break
8:          $TimingRecovery(SF_t, 0, X)$
9:      end for
10:     if no timing violation and power is improved with $current\_sol$
11:         $best\_sol \leftarrow current\_sol$
12:         decrease $th_k$, $X_k$
13:     else
14:         $stuck\_count \leftarrow stuck\_count + 1$
15:         increase $th_k$, $X_k$
16:     end if
17:     $KickMove(th_k, X_k)$
18:     if $stuck\_count > S_{max}$
19:         $SF \leftarrow SF'_p$
20:     end if
21: end for
```

Overall optimization. We have several user-defined input parameters that can tune the optimization flow. Table 6 shows the user-defined input parameters and default values. The details of the overall optimization flow are shown in Algorithm 2. We first store the initial solution to *best_sol* to set it as our starting point (Line 1). We set *SF* to the input SF_p for power recovery procedure (Line 2). We then perform power reduction with *SF*, the trial rate X and th_p (Line 5). We give a small negative slack margin th_p as default to allow aggressive downsizing and Vt swapping for power reduction. We observe from our experiments that optimization with a negative slack margin followed by timing recovery typically achieves better solution quality compared to the conventional zero-margin optimization. After the power reduction step, we iteratively perform the procedures for fixing timing violations until all violations are fixed or $j = T_{max}$. If there is no timing violation with the current solution, and its power is less than the best power value, we store the current solution as the best solution (Lines 10–11). And, th_k and X_k are decreased so that less cells are upsized in the kick move stage (Line 12). If not, we increase the *stuck_count* which indicates that the optimization gets stuck (Line 14). And, we increase th_k and X_k (Line 15) so that more cells are upsized in *KickMove*. We then perform *KickMove* (Line 17). If *stuck_count* is larger than S_{max}, SF is changed to SF'_p (Line 19). We iterate the opti-

[5] This recalls the metaheuristic paradigm of *threshold acceptance;* see, e.g., Ref. [50].

mization loop by $O_{\max}$. In our current implementation, the cells and moves in the *KickMove* procedure are determined by a given SF, th_k and X_k (Algorithm 5). The SF is deterministic, while th_k and X_k change throughout the entire optimization process. And, although we change th_k and X_k throughout the optimization as described in Algorithm 2, the values are not randomly selected. Therefore, our current *KickMove* implementation is not completely stochastic. However, as the parameters th_k and X_k, and the slacks of cells, change throughout multiple iterations, the target cells for Vt swap or sizing will change. The net result is a kick move whereby the current state of the circuit is “perturbed” in a manner designed to escape the current local minimum – sometimes referred to as a *basin of attraction* – and thus avoid cycling in the optimization.

Algorithm 3 Procedure of power reduction.

```
Procedure: $PowerReduction(SF_p, X, th_p)$
Input: sensitivity function $SF_p$, trial rate $X$, slack threshold $th_p$
Output: an updated netlist with less power
1:  for all cell $c \in netlist$ except for clock do
2:      for all move $m \in$ candidate moves $M_c$ do
3:          $sol.cell \leftarrow c$; $sol.move \leftarrow m$
4:          $sol.score \leftarrow CalculateSensitivity(c, m, SF_p)$
5:          if $\Delta P < 0$ then $sol\_list \leftarrow sol\_list \bigcup \{sol\}$
6:      end for
7:  end for
8:  sort $sol\_list$ in order of increasing sensitivity score
9:  $cnt \leftarrow 0$
10: while $cnt < X$ do
11:     if any prior move exists in top $PR_{\max}$ solutions then
12:         $sol \leftarrow PickFirstPriorMove(sol\_list)$; $PR_{\max} \leftarrow PR_{\max} - 1$
13:     else
14:         $sol \leftarrow sol\_list.front()$
15:     end if
16:     commit $sol$
17:     $sol\_list \leftarrow sol\_list \setminus \{sol\}$
18:     if $\mathrm{mod}(cnt, \delta) = 0$ then $TimingCorr()$
19:     if $s_{sol.cell} < th_p$ or new MaxTran/MaxCap occurs in any
        timing view then
20:         revert $sol$
21:     else
22:         $cnt \leftarrow cnt + 1$
23:     end if
24:     if $worst\_slack < th_p$ then break
25: end While
```

Power reduction. Algorithm 3 illustrates the sensitivity-based power reduction procedure (*PowerReduction*). The inputs to *PowerReduction* include the sensitivity function SF_p, trial rate X and the slack threshold th_p. We first build a list of candidate solutions where a solution *sol* indicates a (target cell *sol.cell*, target move *sol.move*) pair, together with its sensitivity score *sol.score*. We calculate the sensitivity score for each (cell, move) pair (Line 4). If ΔP of the candidate (cell, move) pair is negative (indicating that power decreases), we add the solution to the solution list *sol_list* (Line 5). As we downsize or increase Vt, the delay of the target cell increases while the power decreases in most cases. Thus, the sensitivity score is typically a negative value, where a smaller value (i.e., a larger absolute value) indicates a better solution. Therefore, the solution list is sorted in increasing order of sensitivity scores (Line 8). The only case where the sensitivity score can be positive is when both delay and power decrease. In this case, we set the sensitivity score to the negative infinite number so that we can always pick such moves first. For details, please see Table 5 in Section 4.2.

We attempt each candidate move in the solution list (Lines 10–25). To prioritize Vt swapping (resp. sizing) for leakage (resp. dynamic) power optimization (as described in Section 4.5), we search prioritized moves among the first $PR_{\max}$ solutions in the solution list (Lines 11–12). In a leakage-only optimization, if we find a Vt-swapping move, we pick the corresponding solution (Line 12), otherwise we pick the first solution among *sol_list* (Lines 13–14). We then commit the move (Line 16). The committed move is removed from the solution list (Line 17). We then update design timing and check the updated slack of the optimized cell in all timing views. During the timing update process, we periodically correlate timing with a signoff timer for accuracy (Line 18). We revert the move if there is any setup timing (i.e., with respect to th_p), MaxTran or MaxCap violation (Lines 19–20). Even though we check timing for every move to avoid violations, the mismatch between the internal timer and the signoff timer can result in timing violations during our optimization. In this case, we quit the procedure (Line 24) and call the *TimingRecovery* procedure. The trials for sizing or Vt-swapping continue until the number of trials reaches X (Line 10)

Algorithm 4 Procedure of timing recovery.

```
Procedure: $TimingRecovery(SF_t, th, X)$
Input: sensitivity function $SF_t$, slack threshold $th$, trial rate $X$
Output: an updated netlist with an improved worst slack
1:  if MaxCap violations exist then FixMaxCap()
2:  if MaxTran violations exist then FixMaxTran()
3:  for all cell $c \in netlist$ except for clock do
4:      if $s_c > th$ continue//skip this cell
5:      for all move $m \in$ candidate moves $M_c$ do
6:          $sol.cell \leftarrow c$; $sol.move \leftarrow m$
7:          $sol.score \leftarrow CalculateSensitivity(c, m, SF_t)$
8:          if $\Delta d_{sol.cell} < 0$ then $sol\_list \leftarrow sol\_list \bigcup \{sol\}$
9:      end for
10: end for
11: sort $sol\_list$ in order of decreasing sensitivity score
12: $cnt \leftarrow 0$
13: while $cnt < X$ do
14:     $sol \leftarrow sol\_list.first()$
15:     $orig\_slack \leftarrow s_{sol.cell}$
16:     commit $sol$
17:     $sol\_list \leftarrow sol\_list \setminus \{sol\}$
18:     if $\mathrm{mod}(cnt, \delta) = 0$ then $TimingCorr()$
19:     if $s_{sol.cell} < orig\_slack$ or creates hold violations then
20:         revert $sol$
21:     else
22:         $cnt \leftarrow cnt + 1$
23:     end if
24: end While
```

Timing recovery. Algorithm 4 illustrates the *TimingRecovery* procedure. In the *TimingRecovery* procedure, we upsize or swap a cell to a lower-Vt cell to fix timing violations caused during the *PowerReduction* procedure. Fixing MaxTran and MaxCap violations is performed first (Lines 1–2). We fix MaxTran and MaxCap violations before performing the SF-based upsizing/Vt-swapping, since we observe that fixing maximum transition and capacitance violations also helps to fix setup time violations by improving gate delays in our experiments. This is because large transitions and/or load capacitance tend to increase gate delays. We also note that it is recommended to first fix electrical rule violations (e.g., maximum transition and capacitance violations) in the DAC Knowledge Center article of MacDonald [51]. For each non-clock cells with slacks less than *th* (Lines 3–4), we calculate sensitivity scores (Lines 6–7) for its candidate moves (i.e., one-step upsizing or one-step decreasing Vt). If the $\Delta d_{sol.cell}$ decreases, we add *sol* to *sol_list*. We then sort the solution list in decreasing order of sensitivity scores (Line 11). The delay terms (i.e., Δd_i, $-\Delta s_i$) are negative values, and the ΔP term is a positive value since delay decreases and power increases with

Table 7
Timing view definitions.

Lib.	View	Mode	Proc.	Volt. (V)	Temp. (C)	RC
140 nm	V1	Func	SS	1.1	150	Cmax
	V2	Test	SS	1.1	150	Cmax
	V3	Func	SS	1.1	−40	Cmax
	V4	Test	SS	1.1	−40	Cmax
65 nm	V1	Func	SS	0.9	125	Cmax
	V2	Func	TT	1.0	25	Cmax
	V3	Func	FF	1.1	125	Cmax
	V4	Func	FF	1.1	−40	Cmax
28 nm	V1	Func	SS	0.9	125	Cmax
	V2	Func	SS	0.9	−40	Cmax
	V3	Func	SS	1.1	125	Cmax
	V4	Func	SS	1.1	−40	Cmax

upsizing or decreasing Vt, in most cases. As a result, the sensitivity scores for timing recovery are negative values, and the solutions with larger sensitivity scores (i.e., smaller absolute values) should have higher priorities. For each solution in the solution list, we commit the corresponding move and update timing (Line 16). During the procedure, we periodically correlate timing with a signoff timer for accuracy, similarly as in the power reduction procedure (Line 18). We then check whether the committed move degrades slack and creates a hold violation in the given timing view v (Line 19). If so, we revert the move (Line 20).

Kick move. Algorithm 5 illustrates the *KickMove* procedure. The purpose of *KickMove* is to perturb the current status so that the optimization does not get stuck at local minima. Thus, for the *KickMove* procedure, we perform upsizing on cells that have positive slacks but less than th_k, with a sensitivity function that considers only timing (Line 1). The number of cell moves during the *KickMove* is limited by X_k. The default number of X_k is set to 0.05% of the total number of cells so as not to allow overly aggressive perturbation.

```
Algorithm 5  Procedure of kick move.
Procedure: $KickMove(th_k, X_k)$
Input: slack threshold for kick move $th_k$, maximum #trials $X_k$
Output: an updated netlist
1:  $SF \leftarrow 1/(\Delta d_i \cdot \#paths_i)$
2:  $TimingRecovery(SF, th_k, X_k)$
```

6. Experimental setups and results

Sizer is implemented with C++ and a Tcl socket interface [52] to communicate with golden signoff timers. Sizer supports the standard.spef/.v formats by using *OpenAccess (OA) 22.43* [9] API. We extend the Liberty parser provided by ISPD-2013 contest to support general Liberty (.lib) files and collect internal power information of cells. The extended Liberty parser is validated with various foundry libraries, i.e., 140 nm, 65 nm GP and 28 nm LP.

6.1. Experimental setup

Testcases. We use five testcases in our experiments, each of which is implemented with two foundry technologies – 65 nm GP with triple-Vt libraries and 28 nm LP with dual-Vt libraries. To validate solution quality as well as scalability of Sizer, we use one design (*aes*) from *OpenCores* website [53], a simplified version of ARM Cortex M0 (*m0*), and testcases with three and five copies of the M0 (i.e., *m0x3*, *m0x5*) as well as matrix multiplier (*mmult*) from the ISPD-2013 benchmark suite. To implement *m0x3* and *m0x5*, we connect primary inputs and outputs of *m0* with muxes at the RTL level. We synthesize from RTL to gate-level netlist using *Synopsys Design Compiler H-2013.03-SP3* [54] for *aes*, *m0*, *m0x3* and *m0x5*. For *mmult*, we map the gate-level netlist from ISPD testcases to given technologies using *Synopsys Design Compiler H-2013.03-SP3* [54]. To perform placement and routing (P&R), we use *Cadence Encounter Digital Implementation System XL 14.2* [11]. We further optimize the designs using a leading commercial tool, namely, *C1*, with high-effort leakage and dynamic power optimizations. For golden signoff timers, we use *Synopsys PrimeTime J-2014.12* [5] (PT) and *Cadence Tempus 14.2* [4] (TMP). The tool versions represent the most up-to-date common flow that could be realized across multiple organizations when this work was performed (see Section 6.5). We do not map specific results to specific tools (although we do indicate a “universe” of tools that we have used), in order to avoid any reporting that could possibly be construed to be “benchmarking”.

We also validate Sizer on a design from *NXP Semiconductors* [13] implemented with a 140 nm foundry technology (dual-Vt), which we refer to as *NXPIC*. The design is used in an application of RFCMOS for the keyless entry/go system of automobiles. We apply our Sizer to one of the blocks of NXP's design, which contains ∼140 K standard-cell instances.

MCMM implementation. To implement the designs in the MCMM context, we define multiple views with various PVT corners and function modes. Table 7 summarizes function modes (Column 3), PVT corners (Columns 4–6) and wire RC corner (Column 7) for each view in three foundry technologies. We use four views selected from available libraries in 65 nm and 28 nm foundry technologies. For 140 nm foundry technology, we use four representative views (Rows 2–5 of Table 7) selected from among 35 views with which the *NXPIC* is implemented.[6] We experimentally confirm that the selected four views are the dominant views among the 35 views based on timing criticality. Ideally, considering all timing views in sizing optimization will produce more robust results that honor timing constraints in every timing view. However, the computation costs increases in proportion to the number of timing views. Thus, the timing views to consider in optimization must be carefully selected to reduce the computational cost. In particular, a timing view having sufficiently large positive slack on its worst timing path will typically not be a dominant view. In the NXPIC case, we exclude the timing views with more than 20 × of the worst (positive) slack throughout all the timing views. For Example, since the worst initial slack is 0.033ns in V1 (Table 8), we exclude the timing views with slack values > 0.66 ns. The timing information in Table 8 is reported by TMP. We do not include TT and FF corners as our optimization corners since (i) setup time violations are dominant in the testcases in our experiments; (ii) downsizing/increasing Vt usually creates setup time violations, and slow corners are more critical during our optimization; and (iii) as we do not touch cells on clock trees, extra (hold) violations are not created at non-slow corners, i.e., TT and FF corners. We experimentally confirm that no extra hold violations are created in optimized designs. In particular, the optimized NXPIC in Table 15 does not have any hold or setup violations in the 31 views other than the selected four views. The detailed information of the testcases is summarized in Table 8. For 65 nm and 28 nm designs, we set the clock period of each view to a view-specific value in order to avoid the situation that a particular view dominates others. Thus, the difference in initial slack values between views is less than or equal to 200 ps The timing information in Table 8 is reported by TMP. For hold slack, we report the worst slack and the sum of total negative slack for the four target views.

[6] The 35 timing views are the combinations of three functional modes and three voltages, three process corners (SS, TT and FF) associated with three temperature corners ($-40°C$, $25°C$ and $150°C$) with complementary views ($\#modes \times \#voltages \times \#corners + \#complementary\ views = 3 \times 3 \times 3 + 8 = 35$).

Table 8
Summary of testcases. The designs are optimized with high-effort optimization options for leakage and dynamic power using *C1*. The values are reported by TMP.

Lib.	Design	#Cells	Clock Period (ns)				Worst Setup Slack (ns)				Hold Slack (ns)		Power (mW)	
			V1	V2	V3	V4	V1	V2	V3	V4	WNS	TNS	Leakage	Total
140 nm	NXPIC	140 K	NA	NA	NA	NA	0.033	0.104	0.264	0.574	NA	NA	4.17e-5	NA
65 nm	m0	13139	1.60	1.10	0.80	0.80	−0.003	0.066	0.02	0.046	−0.294	−39	0.40	20.14
	aes	20583	0.90	0.60	0.40	0.40	−0.014	0.032	−0.005	0.013	−0.144	−82	0.78	112.50
	m0x3	35473	1.85	1.15	0.85	0.85	0.001	0.003	0.003	0.035	−0.347	−54	0.86	50.09
	m0x5	55160	1.90	1.20	0.90	0.90	−0.012	−0.017	−0.005	0.03	−0.185	−54	1.23	74.87
	mmult	123283	2.10	1.40	1.00	1.00	0.002	0.05	0.011	0.043	−0.008	0	3.47	191.98
28 nm	m0	9964	1.40	1.60	1.00	1.00	0.016	−0.010	0.036	0.013	−0.276	−44	0.16	15.27
	aes	13154	0.90	1.10	0.60	0.60	0.010	0.023	0.011	−0.005	−0.113	−57	0.25	56.76
	m0x3	30155	1.40	1.70	1.00	1.00	−0.273	−0.313	−0.139	−0.183	−0.417	−165	0.79	80.13
	m0x5	50231	1.50	1.85	1.25	1.25	−0.013	−0.017	0.184	0.153	−0.303	−100	0.47	58.77
	mmult	107619	1.80	2.10	1.30	1.30	−0.011	−0.017	0.037	0.003	0	0	1.66	186.18

Table 9
Design of experiments.

	Tech.	Tool	Signoff	Objective	MCMM	SF
Expt 1	65 nm	*Tri-R* *C2*	PT	Leakage	No	SF3
Expt 2	28 nm 65 nm	*C1*	TMP	Leakage Total power	Yes	MSF1
Expt 3	65 nm	–	TMP	Leakage	No	SF1-6
Expt 4	140 nm	*C1*	TMP	Leakage	Yes	MSF1-2

Design of Experiment. We have conducted four experiments to demonstrate the performance of Sizer.

- **Expt 1** shows the comparison between Sizer, a reproduced version of [3] and a commercial leakage optimization tool, namely, *C2*.
- **Expt 2** shows our optimization results with MCMM on the optimized designs with high effort option using *C1*.
- **Expt 3** studies the impact of six SFs on the solutions.
- **Expt 4** shows the optimization results of a commercial product design in 140 nm.

Table 9 summarizes the libraries, tools, signoff tools, the objectives of optimization, whether MCMM is considered, and the sensitivity functions used for each experiment.

6.2. Expt 1: Comparison between Sizer [3], and a commercial tool (C2)

In Expt 1, we compare the performance of our Sizer against our reproduced version of Trident [3] (*Tri-R*) and *C2* on contest benchmarks and five testcases implemented at 65 nm foundry technology.

***Tri-R*, reproduced Trident**. As Trident [3] is designed to perform for contest testcases, it does not support inputs with standard formats.

Table 10
Reproduction of ISPD Trident (Tri) results with Tri-R, and comparison with Sizer (Szr). The values are reported by PT.

Design	#Cells	Leakage (mW)				Runtime (min)		
		Init	Tri	Tri-R	Szr	Tri	Tri-R	Szr
usb_phy_fast	608	1.73	1.59	1.60	1.59	0.21	0.99	2.7
usb_phy_slow	608	1.1	1.07	1.07	1.05	0.17	0.64	1.8
pci_bridge32_fast	30603	145.6	101.90	99.82	113.24	12	23.62	68
pci_bridge32_slow	30603	65.3	58.83	59.04	59.26	5.39	11.95	65
fft_fast	32766	583.04	305.29	305.38	357.01	32.58	116.58	116
fft_slow	32766	128.62	93.10	92.94	99.82	17.4	42.03	69
edit_dist_fast	126665	1040	619.30	613.01	1040	170.6	649.19	1321
edit_dist_slow	126665	63	465.60	464.82	544.09	107.2	337.56	1211

Table 11
Leakage optimization result comparison between Tri-R, Sizer (Szr) and a commercial tool (*C2*). The results are reported by PT.

Design	CP(ns)	WNS(ns)	Leak(mW)	WNS (ns)				ΔLeak				Runtime (min)			
				Tri-R	Szr	Szr-I	C2	Tri-R	Szr	Szr-I	C2	Tri-R	Szr	Szr-I	C2
m0	1.64	0.031	0.40	0.031	0	0.000	0	0%	23%	16%	4%	52	263	44	0
aes	1.0	0.001	0.78	0.001	0	0.000	0	27%	46%	32%	22%	42	200	44	0
m0x3	2.11	0.032	0.87	0.006	0	−0.001	0	32%	54%	47%	30%	99	541	106	0
m0x5	2.16	0.003	1.23	0.003	0	−0.002	0	24%	50%	41%	25%	115	942	126	0
mmult	2.15	0.034	3.49	0.034	0	0.000	0	0%	6%	6%	2%	414	1410	391	0

Table 12
Leakage and total power optimization result.

Process	Optimization	Design	Setup Slack (ns)				Hold Slack (ns)		Final Power (mW)		Δ Power		Memory (MB)	Runtime (min)	
			V1	V2	V3	V4	WNS	TNS	Leak	Total	Leak	Total		Total	Timer
65 nm	leakage	m0	0.000	0.052	0.019	0.044	−0.294	39	0.37	20.00	6%	1%	1680	345	279
		aes	−0.004	0.034	0.000	0.011	−0.144	81	0.70	111.33	10%	−3%	2148	1068	944
		m0x3	0.000	−0.001	0.003	0.033	−0.347	53	0.77	49.75	10%	1%	3981	2161	1836
		m0x5	−0.012	−0.017	−0.006	0.030	−0.185	54	1.23	74.87	0%	0%	6518	1983	1724
		mmult	0.000	0.050	0.011	0.043	−0.008	0	3.44	191.83	1%	0%	10673	498	397
	total	m0	0.000	0.062	0.021	0.051	−0.294	39	0.39	19.66	3%	2%	1680	559	466
		aes	0.000	0.032	0.000	0.000	−0.144	81	0.78	108.02	0%	4%	2148	1107	972
		m0x3	0.000	0.000	0.004	0.037	−0.347	53	0.80	48.28	7%	4%	3981	2293	1956
		m0x5	−0.012	−0.017	−0.006	0.030	−0.185	54	1.23	74.87	0%	0%	6518	2406	2075
		mmult	0.000	0.046	0.006	0.038	−0.009	0	3.52	190.66	−1%	1%	10673	3960	3339
28 nm	leakage	m0	0.011	−0.012	0.033	0.008	−0.276	43	0.16	15.24	2%	0%	1310	75	48
		aes	0.008	0.02	0.01	−0.006	−0.113	57	0.25	56.76	0%	0%	1775	224	175
		m0x3	−0.274	−0.317	−0.14	−0.185	−0.417	165	0.47	58.77	0%	0%	3928	237	174
		m0x5	−0.03	−0.029	0.168	0.135	−0.305	112	0.78	80.14	2%	0%	6355	521	347
		mmult	−0.011	−0.017	0.037	0.003	0	0	1.66	186.18	0%	0%	10658	2226	1144
	total	m0	0.005	−0.011	0.03	0.005	−0.27	36	0.17	14.77	−3%	3%	1310	107	68
		aes	0.007	0.026	0.001	−0.022	−0.113	43	0.26	56.53	−6%	0%	1774	383	316
		m0x3	−0.261	−0.364	−0.136	−0.178	−0.417	91	0.50	57.41	−8%	2%	3941	406	234
		m0x5	−0.018	−0.027	0.184	0.153	−0.305	82	0.81	76.61	−2%	4%	6355	598	349
		mmult	−0.014	−0.021	0.035	0	0	0	1.66	186.25	0%	0%	10658	2551	1276

Thus, we extend Trident to enable support for the standard.spef/.v/.lib formats. We refer to this extended version of [3] as *Tri-R*. We confirm that Tri-R reproduces similar results to that in Ref. [3] (Table 10). The golden signoff timer is PT to be consistent with the golden timer used in the ISPD contest. The slight difference in the reported leakage results is due to sensitivity to the order of calculation as well as randomness seen in multi-threaded operation. The longer runtime in Tri-R is due to (i) more complex data structures and extra input processing steps (e.g., reading input.spef/.v into OA database); and (ii) avoidance of certain hard-wiring of codes used by Ref. [3] for speedup. One caveat of academic sizers is that in many cases, hard-wired codes are used to achieve high quality of solutions with minimum runtime for a particular set of inputs (i.e., contest testcases). Un-hardwiring leads to runtime increases. Based on the results, below with the assumption that Trident and Tri-R are equivalent, we compare Tri-R and Sizer with designs at 65 nm technology.

We further compare Trident, Tri-R with Sizer. As Sizer is designed to maintain initial timing slack, Sizer cannot handle the contest benchmarks as is, in which every gate is sized to the largest size/lowest Vt and WNS is a huge negative value. Thus, we use intermediate, timing feasible solutions from Trident as the inputs for this experiment. We observe that Sizer produces similar solution qualities for usb_phy_fast, usb_phy_slow and pci_bridge32_slow, but does not perform well for the other contest benchmarks with longer runtimes. This may reveal the gap between academic sizers that are optimized toward academic benchmarks and commercial sizers, which was also noted in the studies of [6].

Comparison of Sizer, *Tri-R* and *C2* with 65 nm designs. Table 11 shows leakage results comparison between Sizer (Szr, Szr-I), Tri-R, and C2. Clock period (CP), worst negative setup slack (WNS), leakage results (Leak) and runtime are reported. The initial/optimized results do not have any hold time violation. Sizer achieves leakage reductions of up to 26% (19% on average) as compared to *Tri-R*. Sizer's runtime is longer than that of *Tri-R*, due to more iterations of optimization. Thus, we also report the intermediate results of Sizer (Szr-I) so as to enable an iso-runtime comparison. For the iso-runtime comparison, our intermediate results also outperform *Tri-R* by up to 17% (12% on average).

We further compare our results with the results from *C2*. Sizer achieves further leakage reduction beyond *C2* results by up to 24% (19% on average). Although the runtime of *C2* is much better (the runtime is less than a minute) than Sizer, these data show that *C2* leaves significant room for further leakage optimization. We further observe that we can pay additional runtime for significant additional power optimization using Sizer.

6.3. Expt 2: Sizer optimization results with MCMM on the solutions of C1

In Expt 2, we show experimental results with MCMM-aware optimization using Sizer, on designs that have been optimized by *C1*. To ensure a fair comparison between *C1* versus Sizer, we perform an iso-TNS and iso-WNS comparison (with respect to setup time constraints), such that our optimization honors the initial total negative slack and worst negative slack of the design, and does not permit further timing slack degradation. We perform the experiments at both 65 nm and 28 nm technologies, with two objectives – leakage power reduction and total power optimizations. The leakage and total power values at V1 view are reported by TMP in Table 12. For all designs, there was no extra hold violation after optimization. The initial timing and power information is reported in Table 8. Sizer is able to achieve up to 10% leakage reduction and 4% total power reduction on initial solutions implemented with high-effort optimization for both leakage and total power reduction using *C1*, for 65 nm designs. For 28 nm designs, Sizer achieves up to 2% leakage reduction and 4% total power reduction, respectively. Our Sizer could not optimize power further for some designs (e.g., no leakage reduction for 65 nm m0x5,

Table 13
Leakage optimization results of 65 nm designs with various SF.

Design	ΔLeakage						Runtime (min)					
	SF1	SF2	SF3	SF4	SF5	SF6	SF1	SF2	SF3	SF4	SF5	SF6
m0	14%	16%	23%	1%	0%	3%	256	143	263	143	99	305
aes	26%	31%	46%	8%	8%	13%	242	149	200	163	103	271
m0x3	45%	51%	54%	4%	10%	16%	404	359	541	124	200	873
m0x5	41%	47%	50%	6%	10%	12%	680	1001	942	387	359	1036
mmult	4%	4%	6%	0%	0%	0%	1838	1470	1410	1568	1171	1330

Table 14
Leakage optimization result of 65 nm designs with various MSF.

Design	ΔLeakage			Runtime (min)		
	MSF1	MSF2	MSF3	MSF1	MSF2	MSF3
m0	7%	6%	0%	62	345	48
aes	6%	10%	0%	216	1068	125
m0x3	4%	10%	0%	360	2161	508
m0x5	0%	0%	0%	313	1983	536
mmult	3%	1%	0%	477	498	384

etc.). Of course, the initial designs are already optimized by *C1*, and hence, there is not much timing slack for power optimization to exploit. Indeed, some designs even have negative initial slack; this implies that *C1* used up all the slacks, and ended up with timing-infeasible solutions.

However, we believe that any improvement that Sizer achieves indicates that there is still room for further optimization of existing leading commercial sizers. And, although numbers might seem to be small, even 1 or 2% of power reduction could be helpful for designs that have tight power constraints [55]. Even if Sizer's performance is "only" very similar to that of leading-edge industry tools, Sizer provides an important new research platform for academic research, as it gives an industry-strength implementation accompanied by an open, full description. We feel that this is a strong contrast to industry tools, details of which are often kept highly confidential by EDA companies. Sizer source codes and scripts are available in Ref. [43].

The larger runtime of multi-corner optimization is mainly due to the timing updates and interface with golden signoff timer (as shown in Column 14, in Table 12).

6.4. Expt 3: Impact of various SFs on the solutions

Expt 3 studies power reductions and runtime of various sensitivity functions. Table 13 shows that sensitivity function SF3 leads to the best solution quality, and that sensitivity function is a key determinant of solution quality and runtime. In general, we observe that the *#paths* parameter does not help to achieve better results for the testcases in our experiments. For Example, SF2 and SF4 show the impact of *#paths*; SF2 (without *#paths*) leads to better results than SF4 (with *#paths*). SFs without the parameter *#paths* (i.e., SF1, SF2 and SF3) offer better results than those with the parameter *#paths* (i.e., SF4, SF5 and SF6). Table 14 shows the leakage and runtime results for the three MSFs. MSF3 does not give any power reduction since MSF3 is also used for timing recovery. Indeed, if the same SF is used for power reduction and timing recovery, it is likely that the same set of gates will be selected for both power reduction and timing recovery. Between MSF1 and MSF2, we see that MSF2 has better performance in terms of power reduction.

6.5. Expt 4: Results of commercial industrial design in 140 nm

In Expt 4, we apply Sizer to an industrial design in 140 nm technology. Our results in Table 15 show that Sizer can achieve 7.1% leakage power reduction beyond an input solution that is well-optimized by commercial P&R tools and ECO optimizations. In this experiment, the integrated use of the signoff timer has large runtime overhead due to the following reasons. First, the signoff is signal integrity (SI)-aware; to perform SI-aware timing analysis, large databases of timing noise information are loaded and the full timing analysis takes several minutes. Second, in our experiments, accurate incremental timing analysis is not available due to a tool limitation. For these reasons, 90% of the total runtime on average is consumed by the signoff timer. Due to the run-

Table 15
Leakage optimization result for NXPIC.

Iter	#Swaps	SF	Final WNS (ns)				ΔLeakage	Runtime (min)
			V1	V2	V3	V4		
1	181	MSF2	0.033	0.104	0.264	0.574	1.44%	417
2	889	MSF2	0.033	0.104	0.264	0.574	4.02%	1045
3	117	MSF2	0.033	0.104	0.032	0.034	4.31%	437
4	120	MSF2	0.033	0.104	0.032	0.034	4.57%	420
5	36	MSF2	0.033	0.104	0.032	0.034	4.67%	330
6	12	MSF2	0.033	0.104	0.032	0.034	4.70%	362
7	155	MSF2	0.033	0.104	0.032	0.034	4.85%	507
8	40	MSF2	0.033	0.104	0.032	0.034	4.96%	395
9	222	MSF2	0.033	0.104	0.032	0.034	5.29%	856
10	201	MSF1	0.033	0.104	0.032	0.034	6.99%	874
11	28	MSF1	0.033	0.104	0.101	0.060	7.10%	480

time issue of the signoff timer, we run Sizer several times with a limited number of cell swaps (i.e., 1000) and a limited number of optimizations (i.e., one). We report the number of cell swaps (second column) and leakage power (eighth column) for each iteration. One iteration consists of both the power reduction and timing recovery steps. After the ninth iteration, we observe that the leakage does not reduce further with MSF2.[7] Changing from MSF2 to MSF1 offers an additional 1.3% reduction.

7. Conclusion

Gate sizing optimization has been well studied, and academic sizers have shown significant power reduction on contest testcases [1,2]. However, in the real world, design complications (hierarchical design, existence of hard macros, multiple clocks), modeling complications (state-dependence, complex slew dependence), timing constraints (MCMM, false paths), and electrical constraints (MaxTran, MaxCap) block the application of academic sizers to industrial designs. More crucially, the real world can reveal that techniques and “best practices” identified by academic research (driven by popular contest benchmarks) may not be usable on industrial designs. Our new academic sizer, *Sizer*, reflects a multi-year evolution from a successful “contest” sizing tool to a tool that can outperform high-effort commercial results for a real industrial IC. This evolution has entailed a near-complete change of techniques as compared to our starting point, *Trident* [3]. We describe techniques that are useful in the academic contest setting but not in the real-world context – as well as new techniques specifically developed for Sizer. We also describe the successful application of Sizer to industrial designs. On a design (i.e., *NXPIC*) that is well-optimized by a leading-edge commercial P&R tool as well as ECO steps, Sizer achieves 7% further leakage reduction without any violation of the given setup/hold and maximum transition constraints in a multi-corner multi-mode context.

Our future works include (i) runtime improvement with better timing recovery; (ii) improved, stochastic optimization by introducing different SFs in *KickMove* function and a random parameter; (iii) considerations of other important constraints in industry designs such as noise.

Acknowledgments

We deeply thank Dr. Jin Hu, Dr. Myung-Chul Kim, Prof. Seokhyeong Kang, Prof. Igor L. Markov and Mr. Pankit Thapar for their respective works with us during 2012–2013 ICCAD Contest collaborations. We also thank design and R&D teams at NXP Semiconductors for their help in enabling the reported studies.

References

[1] M.M. Ozdal, C. Amin, A. Ayupov, S. Burns, G. Wilke, C. Zhuo, ISPD-2012 discrete cell sizing contest and benchmark suite, in: Proc. ISPD, 2012, pp. 161–164. http://archive.sigda.org/ispd/contests/12/ispd2012_contest.html.
[2] M.M. Ozdal, C. Amin, A. Ayupov, S. Burns, G. Wilke, C. Zhuo, An improved benchmark suite for the ISPD-2013 discrete cell sizing contest, in: Proc. ISPD, 2013, pp. 168–170. http://archive.sigda.org/ispd/contests/13/ispd2013_contest.html.
[3] A.B. Kahng, S. Kang, H. Lee, I.L. Markov, P. Thapar, High-performance gate sizing with a signoff timer, in: Proc. ICCAD, 2013, pp. 450–457.
[4] Cadence Encounter Timing System User’s Manual, http://www.cadence.com.
[5] Synopsys PrimeTime User’s Manual, http://www.synopsys.com.
[6] A.B. Kahng, H. Lee, J. Li, Horizontal benchmark extension for improved assessment of physical CAD research, in: Proc. GLSVLSI, 2014, pp. 27–32.
[7] Yosys Open Synthesis Suite, http://www.clifford.at/yosys/.
[8] LEF/DEF reference, https://projects.si2.org/openeda.si2.org/projects/lefdef/.
[9] Si2 OpenAccess, http://www.si2.org/?page=69.
[10] Gary Smith EDA, http://www.garysmitheda.com/.
[11] Cadence Encounter Digital Implementation/Innovus User’s Manual, http://www.cadence.com.
[12] Tela Innovations, http://www.tela-inc.com/.
[13] http://www.nxp.com.
[14] M.R.C.M. Berkelaar, J.A.G. Jess, Gate sizing in MOS digital circuits with linear programming, in: Proc. EURO-DAC, 1990, pp. 217–221.
[15] D.G. Connery, K. Keutzer, Linear programming for sizing, V_{th} and V_{dd} assignment, in: Proc. ISLPED, 2005, pp. 149–154.
[16] K. Jeong, A.B. Kahng, H. Yao, Revisiting the linear programming framework for leakage power vs. Performance optimization, in: Proc. ISQED, 2009, pp. 127–134.
[17] C.-P. Chen, C. Chi, D.F. Wong, Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation, IEEE Trans. CAD 18 (7) (1999) 1014–1025.
[18] H. Chou, Y.-H. Wang, C.C.-P. Chen, Fast and effective gate sizing with multiple-V_t assignment using generalized Lagrangian relaxation, in: Proc. ASP-DAC, 2005, pp. 381–386.
[19] Y.L. Huang, J. Hu, W. Shi, Lagrangian relaxation for gate implementation selection, in: Proc. ISPD, 2011, pp. 167–174.
[20] G. Flach, T. Reimann, G. Posser, M. Johann, R. Reis, Effective method for simultaneous gate sizing and Vth assignment using Lagrangian relaxation, IEEE Trans. CAD 33 (4) (2014) 546–557.
[21] L. Li, P. Kang, Y. Lu, H. Zhou, An efficient algorithm for library-based cell-type selection in high-performance low-power designs, in: Proc. ICCAD, 2012, pp. 226–232.
[22] V.S. Livramento, C. Guth, J.L. Güntzel, M.O. Johann, A hybrid technique for discrete gate sizing based on Lagrangian relaxation, ACM TODAES 19 (4) (2014) 40.
[23] M.M. Ozdal, S. Burns, J. Hu, Gate sizing and device technology selection algorithms for high-performance industrial designs, in: Proc. ICCAD, 2011, pp. 724–731.
[24] M.M. Ozdal, S. Burns, J. Hu, Algorithm for gate sizing and device parameter selection for high-performance designs, IEEE Trans. CAD 31 (10) (2012) 1558–1571.
[25] M. Rahman, H. Tennakoon, C. Sechen, Library-based cell-size selection using extended logical effort, IEEE Trans. CAD 32 (7) (2013) 1086–1099.
[26] T. Reimann, C.C.N. Sze, R. Reis, Gate sizing and threshold voltage assignment for high performance microprocessor designs, in: Proc. ASP-DAC, 2015, pp. 214–219.
[27] S. Roy, D. Liu, J. Um, D.Z. Pan, OSFA: a new paradigm of gate-sizing for power/performance optimizations under multiple operating conditions, in: Proc. DAC, 2015, pp. 1–6.
[28] H. Tennakoon, C. Sechen, Gate sizing using Lagrangian relaxation combined with a fast gradient-based pre-processing step, in: Proc. ICCAD, 2002, pp. 395–402.
[29] J. Xie, C.Y.R. Chen, Lookup table based discrete gate sizing for delay minimization with modified elmore delay model, in: Proc. GLSVLSI, 2015, pp. 361–366.
[30] S. Hu, M. Ketkar, J. Hu, Gate sizing for cell-library-based designs, IEEE Trans. CAD 28 (6) (2009) 818–825.
[31] Y. Liu, J. Hu, A new algorithm for simultaneous gate sizing and threshold voltage assignment, IEEE Trans. CAD 29 (2) (2010) 223–234.
[32] M. Rahman, H. Tennakoon, C. Sechen, Power reduction via near-optimal library-based cell-size selection, in: Proc. DATE, 2011, pp. 867–870.
[33] T. Reimann, G. Posser, G. Flach, M. Johann, R. Reis, Simultaneous gate sizing and Vt assignment using fanin/fanout ratio and simulated annealing, in: Proc. ISCAS, 2013, pp. 2549–2552.
[34] T.-H. Wu, A. Davoodi, PaRS: parallel and near-optimal grid-based cell sizing for library-based design, IEEE Trans. CAD 28 (11) (2009) 1666–1678.
[35] P. Gupta, A.B. Kahng, P. Sharma, D. Sylvester, Gate-length biasing for runtime leakage control, IEEE Trans. CAD 25 (8) (2006) 1475–1485.
[36] J. Hu, A.B. Kahng, S. Kang, M.-C. Kim, I.L. Markov, Sensitivity-guided metaheuristics for accurate discrete gate sizing, in: Proc. ICCAD, 2012, pp. 233–239.
[37] M. Rahman, C. Sechen, Post-synthesis leakage power minimization, in: Proc. DATE, 2012, pp. 99–104.
[38] A. Srivastava, D. Sylvester, D. Blaauw, Power minimization using simultaneous gate sizing, dual-V_{dd} and dual-V_{th} assignment, in: Proc. DAC, 2004, pp. 783–787.
[39] L. Wei, K. Roy, C. Koh, Power minimization by simultaneous dual-V_{th} assignment and gate-sizing, in: Proc. CICC, 2000, pp. 413–416.
[40] J.P. Fishburn, A.E. Dunlop, Tilos: a posynomial programming approach to transistor sizing, in: Proc. ICCAD, 1985, pp. 326–328.
[41] C. Moon, P. Gupta, P. J. Donehue and A. B. Kahng, “Designing a Digital Circuit by Correlating Different Static Timing Analyzers”, U.S. Patent No. 7,823,098, 2010.
[42] A.B. Kahng, S. Kang, H. Lee, S. Nath, J. Wadhwani, Learning-based approximation of interconnect delay and slew in signoff timing tools, in: Proc. SLIP, 2013, pp. 1–8.
[43] UCSD Sizer homepage, https://github.com/abk-openroad/TritonSizer.
[44] A.B. Kahng, New game, new goal posts: a recent history of timing closure, in: Proc. DAC, 2015, pp. 1–6.
[45] P. Grassberger, “Go with the Winners: A General Monte Carlo Strategy”, http://arxiv.org/pdf/cond-mat/0201313v1.pdf.
[46] O. Martin, S.W. Otto, E.W. Felten, Large-step Markov chains for the traveling salesman problem, Complex Syst. 5 (3) (1991) 299–326.
[47] http://vlsicad.ucsd.edu/SIZING/optimizer.html, UCSD SensOpt Leakage Optimizer (A. B. Kahng, S. Kang, 2010-2011).
[48] Physical Synthesis Optimization Senior Staff Engineer, Personal Communication, Qualcomm Technologies Inc., November 2018.
[49] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, 1999.

[7] Several additional iterations were attempted with MSF2, but no improvement was observed.

[50] T.C. Hu, A.B. Kahng, C.W. Tsao, Old bachelor acceptance: a new class of non-monotone threshold accepting methods, ORSA J. Comput. 7 (4) (1995) 417–425.

[51] N.D. MacDonald, Timing Closure in Deep Submicron Designs, DAC.com Knowledge Center Article. March 2010, http://vlsicad.ucsd.edu/DAC15/MACDONALD_TIMINGCLOSURE.pdf.

[52] Tcl/Tk Built-in Socket Commands Manual, http://www.tcl.tk/man/tcl8.4/TclCmd.

[53] OpenCores: Open Source IP-Cores, http://www.opencores.org.

[54] Synopsys Design Compiler User Guide, http://www.synopsys.com.

[55] IBM, Physical Synthesis Optimization Research Staff Member, Personal Communication, October 2015.