Floorplan and Placement Methodology for Improved Energy Reduction in Stacked Power-Domain Design

Kristof Blutman†, Hamed Fatemi†, Andrew B. Kahng‡+, Ajay Kapoor†, Jiajia Li‡ and José Pineda de Gyvez†
+CSE and ‡ECE Departments, UC San Diego, †NXP Semiconductors
{abk, jil150}@ucsd.edu, {kristof.blutman, hamed.fatemi, ajay.kapoor, jose.pineda.de.gyvez}@nxp.com

***Abstract*—Energy and battery lifetime constraints are critical challenges to IC designs. Stacked power-domain implementation, which stacks voltage domains in a design, can effectively improve the power delivery efficiency and thus improve battery lifetime. However, such an approach requires balanced current between different domains across multiple operating scenarios. Furthermore, level shifter insertion (together with shifters' delay impacts), along with placement constraints imposed by power domain regions, can incur power and area penalties. To our knowledge, no existing work performs sub-block-level partitioning optimization for stacked-domain designs. In this paper, we present an optimization framework for stacked-domain designs. Based on an initial placement solution, we apply a flow-based partitioning that is aware of multiple operating scenarios, cell placement, and timing-critical paths to partition cells into two power domains with balanced current and minimized number of inserted level shifters. We further propose heuristics to define regions for each power domain so as to minimize placement perturbation, as well as a dynamic programming-based method to minimize the area cost of power domain generation. In an updated floorplan, we perform matching-based optimization to insert level shifters with minimized wirelength penalty. Overall, our method achieves more than ∼10% and 3X battery lifetime improvements in function and sleep modes, respectively.**

I. Introduction

Energy and battery lifetime constraints induce new and critical challenges to IC designs, especially for mobile and IoT (Internet of Things) applications. To achieve power autonomy in the era of a slowing Moore's law, new low-power techniques must be exploited. While many low-power techniques [9] have concentrated on the circuit side of system design, power management techniques have received growing attention due to the importance of power efficiency. Notably, the misalignment of battery voltages compared to scaled core voltages causes inefficiencies that present significant opportunities for power saving. In order to better align SoC power domain voltages with battery voltages, *stacked power domain* (or *voltage stacking*) has been proposed [20][22][25].

Figure 1 illustrates the basic idea of stacked power domain. A stacked power-domain (or stacked-domain) design connects in series power domains that are connected in parallel in a conventional design.[1] Figure 1 shows that one power domain (i.e., the top domain) is placed over the other (i.e., the bottom domain) to double the voltage and (ideally) halve the current compared to that in a conventional design. More specifically, if the supply voltage of a conventional design is V (i.e., $VDD = V$ and $VSS = 0$), then the $\{VDD, VSS\}$ of the top and bottom domains in the corresponding stacked-domain design are $\{2V,\ V\}$ and $\{V,\ 0\}$, respectively. Moreover, in the ideal case the currents are balanced across the two domains. The stacked-domain scheme provides implicit 2:1 downconversion of external supplies. In light of this, there is no need to employ a bulky supply to generate the supply voltage for

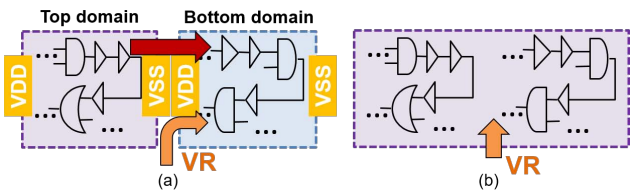

Fig. 1. Comparison between (a) stacked power-domain design, versus (b) conventional design. VR indicates voltage regulator. The orange arrows indicate current from voltage regulators. The red arrow indicates stacked current.

[1]Our study focuses on optimization with two power domains (i.e., top and bottom domains) and conventional 2D implementation (as opposed to three-dimensional integration). Stacked-domain optimization with more than two power domains and/or in 3DICs is left as a direction for future research.

TABLE I
Description of notations used in our discussion.

Term	Meaning
P_{ext}	total input power from external supply (e.g., battery)
$P_{VR,in}$	input power of voltage regulator from external supply
$P_{VR,out}$	output power of voltage regulator to core
η_{VR}	power conversion efficiency of voltage regulator
P_{stk}	direct power of stacked power domain from external supply
P_{core}	total power consumption of core
I_{stk}	stacked current (current from top domain to bottom domain)
I_{VR}	output current from voltage regulator
T	battery lifetime

the core (i.e., gate instances and memories). Instead, it suffices to employ a much smaller converter that acts only as a watchdog to the supply rail that connects the power domains. This results in increased power efficiency for the overall system.

Based on the power conversion modeling proposed in [3], we derive the battery lifetime improvement from stacked-domain optimization as follows. (Table I lists the notations used in our discussion.) Since battery lifetime (T) is inversely proportional to P_{ext}, we compare P_{ext} of a stacked-domain design to that of a conventional design.[2] By definition (see Table I), in a stacked-domain design we have

$$P_{ext} = P_{stk} + P_{VR,in} = P_{stk} + P_{VR,out}/\eta_{VR} \quad (1)$$

On the other hand, in a conventional design, power only outputs from the voltage regulator. Thus, the total input power from the external supply of a conventional design (P'_{ext}) is calculated as

$$P'_{ext} = P_{core}/\eta_{VR} \quad (2)$$

By assuming the same core power consumption in both stacked-domain and conventional designs, we have

$$P_{core} = P_{stk} + P_{VR,out} \quad (3)$$

Furthermore, based on the model described in [3], we have

$$P_{VR,out}/P_{core} = I_{VR}/(2 \cdot I_{stk} + I_{VR}) \quad (4)$$

Finally, based on the above analyses, the battery lifetime ratio between the stacked-domain design (T) versus the conventional design (T') is[3]

$$\frac{T}{T'} = \frac{P'_{ext}}{P_{ext}} = \frac{2 \cdot I_{stk} + I_{VR}}{2 \cdot \eta_{VR} \cdot I_{stk} + I_{VR}} \quad (5)$$

We observe that the battery lifetime benefit from stacked-domain implementation increases with a smaller I_{VR}. As a motivating example, if the current is perfectly balanced between two domains (i.e., $I_{VR} = 0$), assuming $\eta_{VR} = 80\%$, the stacked-domain implementation provides 25% battery lifetime improvement over the conventional implementation. Moreover, since power efficiency of voltage regular decreases with smaller supply current, battery lifetime benefit from stacked-domain implementation is expected

[2]We use battery lifetime (T) as the metric to evaluate energy improvement achieved by our proposed methodology. We also report power values of core (P_{core}) and the entire system (P_{ext}) from our optimization in Table III.

[3]$\frac{P'_{ext}}{P_{ext}} = \frac{P_{core}/\eta_{VR}}{P_{stk}+P_{VR,out}/\eta_{VR}} = \frac{(2 \cdot I_{stk}+I_{VR})/\eta_{VR}}{2 \cdot I_{stk}+I_{VR}/\eta_{VR}}$. See also [3].

to be higher for designs in low-power modes that use a voltage regulator optimized for high-power cases..

Although stacked-domain implementation provides significant battery lifetime improvement, it also raises non-trivial implementation methodology challenges that must be solved. First, the communication between the power domains must be ensured by level shifters that can convert such extreme signal levels. Second, the power efficiency improvement is directly dependent on the current balancing between the two power domains. In other words, the design must be bipartitioned in terms of current consumption. We also note that such a partitioning optimization must comprehend multiple operating scenarios, area and power penalties as well as timing impact of level shifters, as well as additional placement constraints imposed by region definition of power domains. The first challenge has been thoroughly investigated, with and several different level shifter architectures having been proposed [22][25]. However, optimization of partitioning and layout planning of the designs has remained an open challenge that prior works (which have mostly been ad hoc or design-specific) do not ultimately answer for general systems. In this paper, we address this open challenge and provide a comprehensive optimization framework for partitioning and floorplanning of stacked-domain implementation that can be used for a wide range of systems.

The contributions of this paper are as follows.

- We propose a comprehensive optimization framework for stacked-domain implementation. Key elements include a flow-based partitioning with layout and timing-path awareness, heuristics for layout region generation of power domains, and a matching-based optimization for level shifter insertion.
- We are the first to propose a partitioning optimization at the sub-block level for stacked-domain implementation that can be used for a wide range of systems.
- We validate our optimization flow on an industrial design, in the context of an industrial implementation flow. We also compare our method to the recent work of [3].
- Our optimized stacked-power domain designs achieve more than 10% and 3X battery lifetime improvement compared to the conventional designs in function and sleep modes, respectively.

II. Previous Work

In this section, we review the previous literature on (i) stacked-domain implementation and (ii) netlist partitioning.[4]

A. Stacked-Domain Implementation

The circuit blocks needed for a stacked-domain implementation – such as level shifters and voltage regulators – are well-studied in the literature. However, to our best knowledge, no existing work is able to fully automate the implementation flow of a stacked-domain design. Various voltage regulators and level shifters have been studied in [22][25], but the designs used in their studies are quite simple. A smart regulation scheme has been proposed in [20], and the studied design has relatively higher complexity, featuring processor cores. At the same time, in the work of [20] there is no connection between different processor cores, which makes the partitioning problem much simpler. Similarly, [21] and [5] only focus on specific design blocks such as IO cells and memories. A recent work [3] applies stacked-domain optimization to a complete MCU system designed with a standard design flow. The partitioning approach presented in [3] is somewhat ad hoc, and is not applicable to a general design. By contrast, here we present a comprehensive optimization framework for stacked-domain implementation that is applicable to a wider range of designs.

[4]Our stacked-domain optimization problem is different from the power-island generation problem [26][8][13], in that the power-island generation optimization assumes different supply voltages for power domains and minimizes the power overhead from voltage assignments, while our optimization maximizes battery lifetime. Moreover, many critical issues such as timing impact of level shifters, insertion of shifter rows, region definition of power domains, etc. are not addressed in power island-related works.

B. Netlist Partitioning

As a classic problem in VLSI optimization, netlist partitioning has been thoroughly studied in previous literature. A comprehensive, still-relevant taxonomy of approaches is given in [1]. We highlight four basic partitioning approaches.

Move-based approach. To partition a given set of vertices into two partitions with balanced weights and minimized number of hyperedge cuts, Kernighan-Lin [18] and Fiduccia-Mattheyses [11] iteratively move or swap vertices guided by gain functions (within a pass-based structure) to achieve a local optimal solution. This greedy iterative improvement approach is efficient and leads to relatively good solution quality. Important improvements and/or extensions have been proposed, such as multi-way partitioning [16], multi-level extension [6], timing path awareness [15], and "lookahead" gain functions (e.g., gain vectors, CLIP/CDIP and LIFO gain buckets, etc. [10][14][19]).

Mathematical programming-based approach. Other works formulate mathematical programs to optimize netlist partitioning. Shih et al. [24] formulate the timing-aware partitioning problem as quadratic boolean programming. They minimize the total cost of cell-to-partition assignments as well as the number of cuts, with respect to capacity and timing constraints. Goemans et al. [12] use semidefinite programming for partitioning optimization. However, their objective is to maximize the number of cuts under capacity constraints.

Flow-based approach. In light of the max-flow min-cut theorem, Yang et al. [27] propose to use repeated max-flow computations and clustering operations to achieve a balanced bipartitioning solution. The work of [7] documents high efficiency and relatively good solution quality of flow-based partitioning with a min-cut objective.

Clustering approach. Netlist partitioning can also be achieved by bottom-up clustering. For example, Rajaraman et al. [23] propose a clustering approach to minimize the delay from PIs to POs. With a maximum-area constraint for each cluster, they iteratively cluster cells until all cells are clustered.

In this work, we apply the flow-based approach [27] to partition instances into two power domains. We propose several extensions to the existing flow-based partitioning including layout and timing-path awareness, multi-scenario weight (i.e., current) balancing, and a prior clustering step for runtime reduction.

III. Methodology

We now describe our optimization framework for stacked-domain implementation. We first state our stacked-domain optimization problem as follows.

Given: A netlist, timing constraints, level shifters, voltage regulator efficiency, and switching information of instances in the netlist,

Do: partition the netlist instances into two domains, **define** the layout region of each domain, and **place** instances and level shifters, **such that** battery lifetime is maximized.

As implied by Equation (5), to maximize the battery lifetime, our basic objective is to balance the current between the two power domains, while minimizing the power penalty due to level shifter insertion.

Figure 2 shows our overall optimization flow. A common practice in stacked-domain implementation is to partition the netlist (i.e., define the power domain of each instance or block) prior to the floorplanning stage [3]. However, performing power domain assignment before placement can result in suboptimal floorplan and placement solutions. More importantly, placement optimization inserts buffers and sizes cells, which can change the current profile

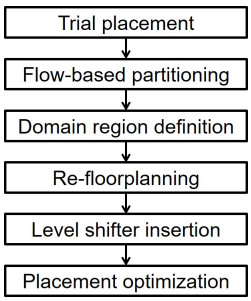

Fig. 2. Overall optimization flow.

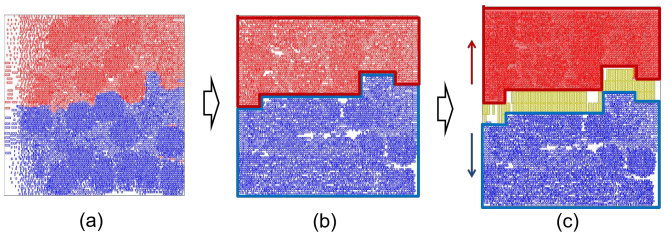

Fig. 3. Example of optimization: (a) layout-aware partitioning, (b) region definition of power domains, and (c) level shifter insertion in the updated floorplan. In blue are instances assigned to the bottom domain and in red are instances assigned to the top domain. In yellow are level shifters. Design: AES (∼11K instances). Technology: 28LP.

of each power domain. As a result, currents that have been balanced during the partitioning stage are no longer balanced after the placement stage. To resolve this, we propose to perform a trial placement, based on which we perform layout-aware partitioning (with minimized number of cuts as well as placement perturbations) to assign instances to power domains. Figure 3(a) shows an example of our layout-aware partitioning solution on design AES [30] in 28LP technology. Based on the partitioning solution, we define the layout region for each power domain such that each domain has a continuous region (Figure 3(b)). Note that since gaps must be inserted along the boundary between two power domains, we propose a dynamic programming optimization to minimize the boundary length between two domains. We then legalize instance placements within the (updated) region for each power domain using a commercial P&R (place-and-route) tool [29]. We then update the floorplan by shifting the power domains (as shown in Figure 3(c)) and inserting level shifters.[5] We perform a matching-based optimization to determine level shifter placement locations that minimize wirelength. Last, we perform an incremental placement optimization to fix timing violations.

A. Flow-Based Netlist Partitioning

The greedy iterative partitioning approach is not naturally amenable to timing path-aware partitioning, and has no mechanism to preserve solution structure of an initial (trial) placement. And, mathematical programming-based approaches typically have large runtimes. Thus, we apply the flow-based approach described in [27] to partition instances into two power domains. Figure 4 illustrates the basic idea of the flow-based partitioning. According to the max-flow min-cut theorem, the approach finds the partitioning solution with the minimum number of cuts for a given netlist via a max-flow optimization. However, this does not guarantee that the balancing constraint is met. To address this, after each max-flow optimization, the approach clusters the vertices belonging to the smaller partition

[5]We understand that modification of the block size might not be consistent with certain implementation flows. At the same time, we believe that performing initial trial placement (with appropriate instance bloating) in a block having Figure 3(c)'s shape will not diverge significantly from the initial trial placement in Figure 3(a), particularly with improved (smaller) level shifter designs. Ongoing work is aimed at a predictive (or, "one loop") methodology to determine the block size prior to trial placement.

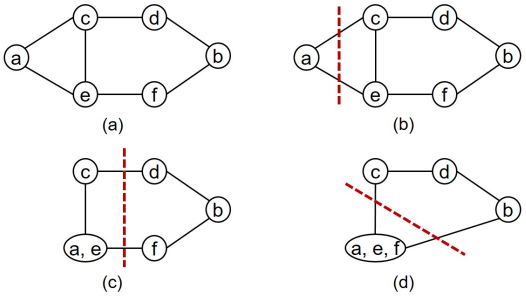

Fig. 4. Flow-based partitioning. a and b are source and sink, respectively. All vertices have the same weight. Red dotted lines indicate cuts. (a) Initial flow network. (b) First max-flow min-cut computation. (b) Clustering operation. (c) Second max-flow min-cut computation.

together with one neighbor vertex (to avoid obtaining the same partitioning solution) into one super vertex. Based on the updated flow network, another max-flow optimization is performed. The approach iteratively performs (incremental) max-flow optimization and clustering until the balancing constraint is satisfied.

We adopt the flow-based partitioning approach to our stacked-domain optimization with the following five extensions.

Source and sink selection. The approach in [27] randomly picks two nodes (instances) in the flow network (netlist) as the source and sink nodes. However, there are cases in which the flows between selected source and sink vertices cannot cover the entire flow network, resulting in unbalanced partitioning solutions. As an example, selection of vertices a and b as the source and sink from the flow network shown in Figure 5(a) will not able to achieve a balanced partitioning solution. To address this, we add a super source and a super sink and connect them to multiple vertices (e.g., PIs and POs in a netlist, or instances located at the core boundary) with edges of infinite capacity to minimize the number of uncovered vertices (instances) as shown in Figure 5(b).

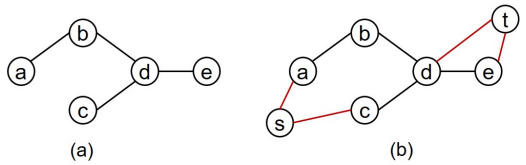

Fig. 5. (a) Choose a / b, or c / d, or d / c as source / sink cannot lead to a balanced solution. (b) Adding a super source (s) and a super sink (t) resolves the issue. Edges in black have unit capacities. Edges in red have infinite capacities.

Layout awareness. To avoid excessive placement perturbation, which can result in current profile change and power penalty, the partitioning optimization must be aware of trial placement locations of instances – such that instances partitioned into the same power domain are placed close to each other in the original trial placement. We achieve this required layout awareness in two ways. (i) We only select the instances located close to each other to connect to the super source (or super sink). As an example, we select instances located within a particular distance (e.g., ten cell rows) from the bottom (resp. top) core boundary to connect to the super source (resp. super sink). (ii) After each max-flow optimization, we detect *outliers*, which are instances belonging to the larger-current partition that are located within a region with a majority of instances belonging to the smaller-current partition. We then cluster these outliers with the instances from the smaller-current partition.

Critical-path awareness. Ignoring signal flow direction and timing path structure during the partitioning optimization can easily result in multiple cuts along one timing path. We extend the partitioning flow in [27] to minimize the number of cuts along timing-critical paths. Similarly to the layout awareness extension discussed above, after each max-flow optimization we detect "V-shaped vertices" [15], which are a sequence of instances belonging to the

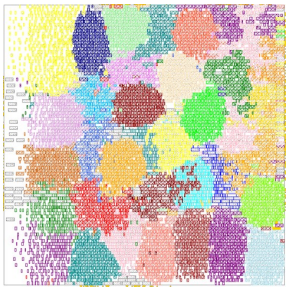

Fig. 6. HEM clustering solution. Different clusters are indicated by different colors. (But since we are limited by 64 available colors, different clusters can have the same color. Also, clusters with small size might not be visible from the figure.) #Clusters = 200. Levels of clustering = 18. Clustering ratio at each level = 0.76. Design: AES. Technology: 28LP.

larger-current partition along a timing-critical path, where the fanin and fanout instances of these instances along the timing-critical path are in the smaller-current partition. We then collapse (cluster) the instances corresponding to the V-shaped vertices into the smaller-current partition without violating the balancing constraints.

Pre-clustering. Although the max-flow optimization can be achieved with an incremental flow computation and the entire optimization takes $O(N)$ iterations to converge where N is the number of instances in a design, the runtime in practice can be substantial for a large design. To reduce the runtime, we perform a pre-clustering optimization based on the heavy-edge matching (HEM) strategy [17]. We enforce layout-awareness constraints (i.e., an upper bound on the distance between two vertices that can be clustered) during the HEM. Figure 6 shows an example of the HEM clustering up through 18 levels (clustering ratio = 0.76 at each level), showing how instances within the same cluster are spatially proximate. Our experimental results show that we can reduce runtime by 75% (two HEM levels and overall clustering ratio of 0.5) with negligible degradation of solution quality (e.g., cut number).

Multiple operating scenarios. To ensure high power efficiency across different operating scenarios, the partitioning optimization must balance currents between two domains across multiple scenarios. To achieve this, we use the weighted sum of normalized currents from different scenarios during our optimization. Specifically, the delta current is calculated as

$$\Delta I = \sum_i (w^i \cdot |I^i_{top} - I^i_{bot}| \ / \ (I^i_{top} + I^i_{bot})) \tag{6}$$

where I^i_{top} and I^i_{bot} are respectively the currents of top and bottom domains in the i^{th} mode, and w^i is the weighting factor of the i^{th} mode, such that $\sum_i w^i = 1$. Our optimization ensures that ΔI does not exceed a predefined upper bound.

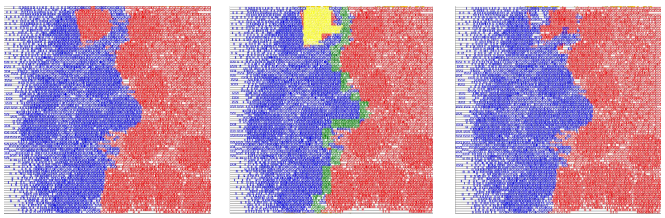

Fig. 7. FM-based grid movement. (a) Initial placement solution. In red and blue are instances partitioned to top and bottom domains. (b) In yellow are outliers of the top domain. In green are neighboring grids of the top domain. (c) Post-movement placement, where each domain has a continuous region. (The small number of remaining outliers are minority instances in their grids, and will be legalized during an incremental placement.) Design: AES. Technology: 28LP.

B. Domain Region Definition

In this section, we describe our methodologies to define the layout region (power island) for each power domain. The definition of the layout region for each power domain affects the design quality in two fundamental ways. (i) Gap area must be inserted along the boundary between different power domains. Therefore, a longer boundary length will lead to higher area penalty. (ii) The power domain definitions will have downstream impact on the PDN (power delivery network) design, which is not yet implemented at this point. Therefore, it is desirable to adjust the power domain definitions for minimized area, power and performance penalties. If the partitioning and the trial placement results in discontinuous power domains, the length of the power domain boundaries is highly likely to be longer compared to the case when the regions of each power domain are merged. Moreover, the power routing will be more difficult, since different power rails will need to be routed to discontinuous power domain regions. Thus, we seek to have only two regions (i.e., power islands) corresponding to the two power domains.

Although our partitioning optimization is layout-aware, there can still be separated regions for each power domain. Figure 7(a) shows an example trial placement and partitioning solution where the top domain (shown in red) has two separated regions. To merge the regions while minimizing placement perturbation (e.g., wirelength increase), we perform an FM-based grid movement optimization (i.e., an iterative, swap-based greedy algorithm as described in Algorithm 1). We first divide the core area into grids. The power domain of each grid if defined as the power domain of majority instances within the grid. We then define the outliers (i.e., grids outside the largest continuous region of the corresponding domain) and neighboring grids (i.e., grids adjacent to the largest continuous region of the different domain) (Line 1). Figure 7(b) shows an example of outliers and neighboring grids. We calculate the cost to swap pairs of outliers and neighboring grids (Lines 2-8). We iteratively swap the pair of an outlier and a neighboring grid with the minimum movement cost, until all outliers (e.g., yellow grids in Figure 7(b)) are removed (Lines 9-16).

Algorithm 1 FM-based grid movement.

```
1:  U ← find outliers; H ← find neighboring grids
2:  for all u ∈ U, h ∈ H do
3:    if u.domain ≠ h.domain then
4:      cost(u, h) ← HPWL increase by swapping u and h
5:    else
6:      cost(u, h) ← +∞
7:    end if
8:  end for
9:  while U ≠ ∅ do
10:   (u', h') ← Min_cost(u,h){(u, h) | u ∈ U, h ∈ H}
11:   swap u' and h'
12:   if create new outliers then
13:     revert the swap; cost(u', h') ← +∞
14:   end if
15:   update U, H and costs
16: end while
```

In the last step of domain region definition, we apply dynamic programming to minimize the length of the boundary between two power domains while maintaining the area within each domain. As the base cases, we calculate the boundary length decrease of each boundary segment by simplifying the boundary shape (e.g., highlighted segment in Figure 8). We note that such simplification must meet an upper bound of moved area (i.e., total area with changed domain assignment). Assuming that the (turning) points along the boundary are indexed from left to right or from bottom to top, the recurrence relation in our dynamic programming optimization is

$$Sol(j) = Min(Sol(i).length + seg(i,j).length), \ \forall 1 < i < j \tag{7}$$

where $Sol(j)$ is the optimized boundary solution from the first point to j^{th} point, and $seg(i, j)$ is the simplified boundary segment

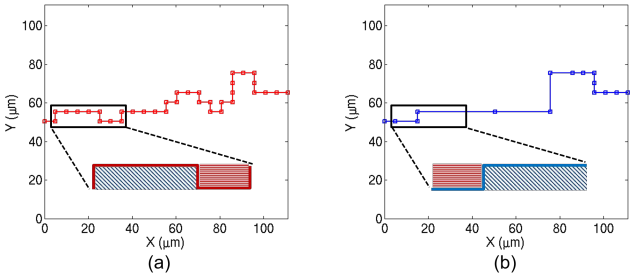

Fig. 8. Boundary optimization. (a) Original boundary between two power domains after grid movement. (b) Optimized boundary with smaller length. An example of segment optimization is shown. Optimized segments have smaller total length while maintaining the same area in each power domain. Design: AES. Technology: 28LP.

between the i^{th} and the j^{th} points. The dynamic programming-based boundary simplification has $O(M^2)$ time complexity, where M is the number of points or segments. The time complexity further decreases to $O(M)$ if we only search a limited range of existing sub-solutions (i.e., i in Equation (7)).

C. Level Shifter Insertion

In the last step of our optimization, we perform re-floorplanning based on the defined power domain regions and insert level shifters. We assume that the layout of the level shifter has already included the boundary of the deep n-well of either or both power domains, and that the edges facing either of the power domains have a standard-cell row structure. As a result, we are able to seamlessly integrate the level shifters with zero minimum distance from standard cells. The row height of our level shifter is 6X of the standard-cell row height.[6] As shown in Figure 3(c), in the updated floorplan, we shift the top power domain by the minimum required number of shifter rows for level shifter insertion and insert level shifter instances between two power domains. We perform a matching optimization (using the Hungarian algorithm [28]) to determine the placement locations of level shifters. More specifically, we enumerate possible placement locations in the shifter rows and calculate the potential cost of placing each level shifter onto each candidate placement location. We define the cost as the total HPWL (half-perimeter wirelength) of nets connected to the level shifter. Based on the cost matrix, we perform matching optimization to assign placement location for each level shifter while minimizing the total cost.

IV. Experimental Results

We perform experiments in a 28nm LP foundry technology with dual-VT libraries. We use four design blocks (AES, DES, JPEG, VGA) from OpenCores [30] as our testcases. Parameters of these four testcases are shown in Table II. The worst-case timing and power analysis view for AES, DES, JPEG and VGA is (SS, 0.95V, 125°C). We synthesize designs using *Synopsys Design Compiler vI-2013.12-SP3* [31] and then place and route using *Cadence Innovus Implementation System v16.1* [29]. We set the placement density at the floorplan stage as 70%, and perform timing and power analyses using *Cadence Innovus Implementation System v16.1*. We also validate our optimization framework on an industrial design that contains dual-core M4 MCU and six memories in a 40nm CMOS foundry technology with HVT-only cells. The worst-case timing and power analysis views for the dual-core M4 design are respectively (SS, 0.99V, 125°C) and (TT, 1.1V, 25°C). We implement the M4 design with Cadence tools. However, we cannot disclose the tool names and versions in the industry collaborator's implementation flow. The level shifter implemented in 40nm technology occupies six standard cell rows × 4μm space. The energy/cycle is 35fJ/cycle. The shifter propagation delay for nominal PVT (TT, 1.1V, 25°C) is 400ps. According to the delay, area and power ratios between the level shifter and the minimum-size inverter in 40nm technology, we generate level shifter models in 28nm LP technology. Figure 9 shows the relation between output current versus the power efficiency of the used voltage regulator, provided by our industry collaborators. Our optimization flow is implemented in C++. Functions used in P&R tools are implemented in Tcl. We conduct our experiments on a 2.5GHz Intel Xeon server.

TABLE II
Testcase parameters.

design	technology	#instances	#flops	clock period
AES	28nm LP	∼11K	530	1.2ns
DES	28nm LP	∼17K	530	1.4ns
JPEG	28nm LP	∼42K	4512	1.6ns
VGA	28nm LP	∼58K	17053	2.0ns
M4 (dual core)	40nm	∼113K	15245	20ns

[6]Our level shifter model is from our industry collaborators.

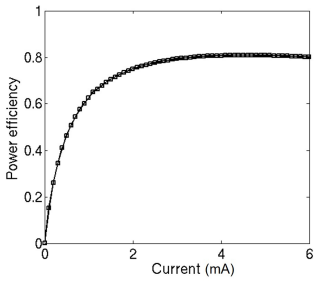

Fig. 9. Power efficiency of switched-capacitor voltage regulator used in [4].

A. Comparison to Conventional Designs

Table III shows the comparison between our stacked-domain optimization (*opt*) versus the conventional implementation (*ref*) on four testcases in 28nm LP and an industrial design in 40nm. Our optimization comprehends both function mode and sleep mode (i.e., with only leakage power). In 28nm technology, our optimization achieves an average of 14% and 238% battery lifetime improvements in function and sleep modes, respectively. On an industrial design in 40nm, we also achieve 11% and 270% battery lifetime improvements in function and sleep modes, respectively. Moreover, the power penalty due to our optimization (see P_{core}) is less than 7% for most cases, while the currents are well-balanced (i.e., with <10% difference) between the top and bottom power domains (see I_{bot} / I_{top}). As a result, our optimization significantly reduces P_{ext}, and leads to an improved battery lifetime. We also observe that the battery lifetime increases more in the sleep mode. This is because the voltage regulator has lower efficiency with smaller current (as shown in Figure 9. Therefore, stacked-domain optimization is expected to provide more energy and battery lifetime benefits in low-power modes if the voltage regulator is optimized for high-power cases. We note that all the implementation solutions except that of [2][3] have negligible timing violations (i.e., #timing violation paths $<$ 5), and the slightly improved worst negative slack (WNS) values of our optimization solutions might be due to tools' noise. Moreover, since logic gates are densely connected in blocks AES, DES, JPEG and VGA, and since the block sizes are small, the relative area overheads due to level shifter insertion are large. Exploration of the tradeoff between power and area penalties due to level shifter insertion versus current balancing is one of our future directions. Runtimes shown in Table III indicate the extra runtime of our optimization that includes partitioning, re-floorplanning and incremental placement optimization. The results also show that simply partitioning two cores into two domains [2] in an evenly partitioned floorplan with a horizontal cut (without considering timing and layout impacts) results in degraded battery lifetime.

B. Sensitivity to Level Shifter Delay

We further study the impact of level shifter model on our stacked-domain optimization. We use a pessimistic (i.e., worst-case) model that has roughly 3-4× power, area and delay compared our current (i.e., nominal-case) model. Figure 10 shows that the pessimistic level shifter model leads to slightly larger total design power (P_{core})

TABLE III
EXPERIMENTAL RESULTS. (POWER UNIT: MW. CURRENT UNIT: MA.)

design	flow	WNS (ps)	#inst	inst area (μm^2)	#LS	func mode				sleep mode				runtime (min)
						P_{core}	I_{bot}/I_{top}	P_{ext}	T	P_{core}	I_{bot}/I_{top}	P_{ext}	T	
AES (28nm)	*ref*	-24	10799	8864	0	8.80	9.26/0.00	10.98	1.00	0.09	0.09/0.00	0.57	1.00	-
	opt	-2	10989	10846	166	9.38	**4.83/5.04**	9.76	**1.13**	0.09	**0.05/0.04**	0.14	**4.19**	8
DES (28nm)	*ref*	4	16505	16790	0	15.83	16.67/0.00	19.76	1.00	0.06	0.07/0.00	0.43	1.00	-
	opt	4	16635	18374	135	16.82	**8.83/8.87**	16.99	**1.16**	0.06	**0.04/0.03**	0.08	**5.16**	7
JPEG (28nm)	*ref*	-4	42131	47486	0	38.54	40.57/0.00	48.12	1.00	0.30	0.32/0.00	0.74	1.00	-
	opt	0	42679	55377	673	40.68	**19.43/23.39**	41.58	**1.16**	0.31	**0.16/0.16**	0.31	**2.39**	14
VGA (28nm)	*ref*	0	57790	97752	0	52.95	55.74/0.00	66.11	1.00	0.35	0.37/0.00	0.86	1.00	-
	opt	2	58165	103824	520	58.28	**28.69/32.65**	59.18	**1.12**	0.35	**0.20/0.17**	0.48	**1.81**	21
M4 (40nm)	*ref*	12	113421	852173	0	7.09	6.45/0.00	8.86	1.00	0.17	0.15/0.00	0.64	1.00	-
	opt	-6	112903	864178	477	7.33	**2.74/3.92**	8.00	**1.11**	0.17	**0.75/0.77**	0.17	**3.70**	34
	[2][3]	-1446	110773	930745	774	8.43	2.46/5.21	9.26	0.96	0.17	0.03/0.14	0.63	1.02	-

due to larger level shifter power and timing impact. Results also show larger ΔI with the pessimistic level shifter model. The larger current difference comes from the level shifters' timing and area impact.

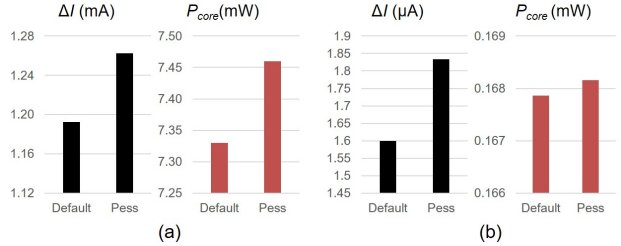

Fig. 10. Impact of level shifter delay, area and power on design QoR in (a) function mode and (b) sleep mode. Design: M4. Technology: 40nm.

C. Sensitivity to Voltage Regulator Power Efficiency

Last, we study impact of voltage regulator efficiency on battery lifetime improvement in stacked-domain designs. More specifically, we assume a higher power efficiency $\eta' = 1 - (\alpha \cdot (1 - \eta))$, where η is the original power efficiency, and (1-α) indicates the efficiency improvement (shown as the x-axis in Figure 11). Figure 11 shows battery lifetime decreases with a higher voltage regulator efficiency, especially in the sleep mode.

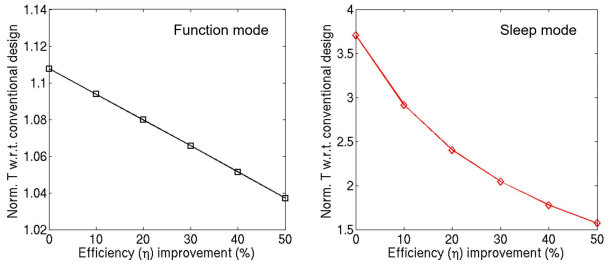

Fig. 11. Impact of voltage regulator efficiency on battery lifetime improvement. Design: M4. Technology: 40nm.

V. CONCLUSION

In this paper, we propose the first comprehensive optimization framework for stacked power-domain implementation with maximized battery lifetime. We extend the existing flow-based partitioning methodology with layout- and timing-path-awareness, as well as multi-scenario balancing objective. We further propose an FM-based grid movement and a dynamic programming-based boundary optimization to define layout region (power island) of each power domain. Last, we insert level shifter rows in an updated floorplan and place level shifters using a matching optimization. We validate our optimization flow in both 28nm LP and 40nm technologies and on an industrial design. Our optimization achieves more than 10% and 3X battery lifetime improvements for function and sleep modes compared to the conventional design. Our future works include (i) a complete IC implementation flow for stacked-domain designs, (ii) a hierarchical and block-aware partitioning optimization, (iii) a predictive methodology to determine the block size prior to trial placement, (iv) stacked-domain optimization with > 2 power domains and/or in 3DICs, and (v) exploration of the tradeoff between power and area penalties that is due to level shifter insertion versus current balancing.

REFERENCES

[1] C. J. Alpert and A. B. Kahng, "Recent Directions in Netlist Partitioning: A Survey", *Integr. VLSI J.* 19(1-2) (1995), pp. 1-81.
[2] K. Blutman, NXP Semiconductors, *personal communication*, June 2016.
[3] K. Blutman, A. Kapoor, J. G. Martinez, H. Fatemi and J. Pineda de Gyvez, "Lower Power by Voltage Stacking: A Fine-grained System Design Approach", *Proc. DAC*, 2016, pp. 78:1-78:5.
[4] K. Blutman, A. Kapoor, A. Majumdar, J. G. Martinez, J. Echeverri, L. Sevat, A. van der Wel, H. Fatemi, J. Pineda de Gyvez and K. Makinwa, "A Microcontroller with 96% Power-Conversion Efficiency using Stacked Voltage Domains", *Proc. IEEE Symposium on VLSI Circuits*, 2016, pp. 1-2.
[5] A. C. Cabe, Z. Qi and M. R. Stan, "Stacking SRAM Banks for Ultra Low Power Standby Mode Operation", *Proc. DAC*, 2010, pp. 699-704.
[6] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Improved Algorithms for Hypergraph Bipartitioning", *Proc. ASP-DAC*, 2000, pp. 661-666.
[7] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Optimal Partitioners and End-Case Placers for Standard-Cell Layout", *IEEE TCAD* 19(11) (2000), pp. 1304-1313.
[8] R. L. S. Ching, E. F. Y. Young, K. C. K. Leung and C. Chu, "Post-Placement Voltage Island Generation", *Proc. ICCAD*, 2006, pp. 641-646.
[9] S. Devadas and S. Malik, "A Survey of Optimization Techniques Targeting Low Power VLSI Circuits", *Proc. DAC*, 1995, pp. 242-247.
[10] S. Dutt and W. Deng, "VLSI Circuit Partitioning by Cluster-Removal Using Iterative Improvement Techniques", *Proc. ICCAD*, 1996, pp. 194-200.
[11] C. M. Fiduccia and R. M. Mattheyses, "Linear Time Heuristic for Improving Network Partitions", *Proc. DAC*, 1982, pp. 175-181.
[12] M. X. Goemans and D. P. Williamson, "Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming", *J. ACM* 42(6) (1995), pp. 1115-1145.
[13] L. Guo, Y. Cai, Q. Zhou and X. Hong, "Logic and Layout Aware Voltage Island Generation for Low Power Design", *Proc. ASP-DAC*, 2007, pp. 666-671.
[14] L. W. Hagen, D. J.-H. Huang and A. B. Kahng, "On Implementation Choices for Iterative Improvement Partitioning Algorithms", *IEEE TCAD* 16(10) (1997), pp. 1199-1205.
[15] A. B. Kahng and X. Xu, "Local Unidirectional Bias for Smooth Cutsize-Delay Tradeoff in Performance-Driven Bipartitioning", *Proc. ISPD*, 2003, pp. 81-86.
[16] G. Karypis and V. Kumar, "Multilevel K-Way Hypergraph Partitioning", *Proc. DAC*, 1999, pp. 343-348.
[17] G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs", *SIAM J. Sci. Comput.* 20(1) (1998), pp. 359-392.
[18] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", *Bell System Tech. J.* 49(2) (1970), pp. 291-307.
[19] B. Krishnamurthy, "An Improved Min-Cut Algorithm for Partitioning VLSI Networks", *IEEE Trans. on Computers* 33(5) (1984), pp. 438-446.
[20] S. K. Lee, T. Tong, X. Zhang, D. Brooks and G. Y. Wei, "A 16-Core Voltage-Stacked System with An Integrated Switched-Capacitor DC-DC Converter", *Proc. Symposium on VLSI Circuits*, 2015, pp. C318-C319.
[21] Y. Liu, P.-H. Hsieh, S. Kim, J. Seo, R. Montoye, L. Chang, J. Tierno and D. Friedman, "A 0.1pJ/b 5-to-10Gb/s Charge-Recycling Stacked Low-Power I/O for On-Chip Signaling in 45nm CMOS SOI", *Proc. ISSCC*, 2013, pp. 400-401.
[22] S. Rajapandian, K. Shepard, P. Hazucha and T. Karnik, "High-Tension Power Delivery: Operating 0.18 μm CMOS Digital Logic at 5.4V", *Proc. ISSCC*, 2005, pp. 298-599.
[23] R. Rajaraman and D. F. Wong, "Optimum Clustering for Delay Minimization", *IEEE TCAD* 14(12) (1995), pp. 1490-1495.
[24] M. Shih and E. S. Kuh, "Quadratic Boolean Programming for Performance-Driven System Partitioning", *Proc. DAC*, 1993, pp. 761-765.
[25] K. Ueda, F. Morishita, S. Okura, L. Okamura, T. Yoshihara and K. Arimoto, "Low-Power On-Chip Charge-Recycling DC-DC Conversion Circuit and System", *IEEE JSSC* 48(11) (2013), pp. 2608-2617.
[26] H. Wu, I.-M. Liu, M. D. F. Wong and Y. Wang, "Post-Placement Voltage Island Generation under Performance Requirement", *Proc. ICCAD*, 2005, pp. 309-316.
[27] H. H. Yang and D. F. Wong, "Efficient Network Flow Based Min-Cut Balanced Partitioning", *IEEE TCAD* 15(12) (1996), pp. 1533-1540.
[28] Hungarian Algorithm, *http://www.informatik.uni-freiburg.de/stachnis/index.html*
[29] Cadence Innovus User Guide.
[30] OpenCores. *http://opencores.org*
[31] Synopsys Design Compiler User's Manual.