**SLIDE: Title**

Thank you for being my committee professors. My topic for today’s presentation is “Improved physical design and signoff methodologies for better design quality”.

**SLIDE: Publications**

This slide shows my publications. In red are the papers I will cover in today’s talk.

**SLIDE: Publications**

This slide shows my publications and posters.

**SLIDE: Challenges in Existing Physical Design/Signoff**

First of all, this slide summarizes the future challenges in physical design and signoff.

Due to small geometries and manufacturing-induced design rules, physical design becomes difficult in advanced nodes. One example is that placement and routing optimization must comprehend pin-accessibility issue.

As we are towards the end of CMOS scaling, design-based equivalent scaling is important. However, this will increase the design complexity and make physical design difficult. Some examples are three-dimensional integration, approximate computing and resilient designs.

In addition, corner explosion is another design challenge. More specifically, we have operating modes such as turbo mode and nominal mode. We have device corners and interconnect corners.

Driven by IoT applications, there is low-power requirement in IC designs. The low-power techniques such as multiple power domain, clock gating, power gating also increase complexity and difficulty in physical design.

More importantly, unable to address these future challenges will lead to degraded design quality.

**SLIDE: In This Presentation**

In my works, I pursue improved physical design optimization and signoff to achieve better tradeoffs among performance, power and area in future VLSI designs.

In this presentation, I will cover three topics.

I will first describe a novel mixed cell-height implementation in advanced nodes.

Then, I will describe min-cost resilient design implementation.

Last, I will show my work on design-stage optimizations for mix-and-match die stacking in 3DICs.

**SLIDE: Mixed Cell-Height Placement in Advanced Nodes**

I will first describe our work of mixed cell-height placement in advanced node.

**SLIDE: Mixed Cell Height Implementation (!)**

In advanced nodes, cells are designed with different heights. For example, cells have different numbers of fins in FinFET technology. Larger cell height offers better timing, but at the cost of area and power. Small cell height has smaller area and power per gate, but has large delay and can result in large number of buffers for a given performance target.

*Unlike conventional device, in FinFET, the conducting channel is wrapped by a thin silicon “fin”. The wrap-around gate structure provides a better electrical control over the channel, and thus offers reduced leakage current.*

This figure shows the area-delay tradeoff of buffers and inverters at 28LP technology. In red are 12T cells and in blue are 8T cells. We can clearly see they that have different area-delay tradeoffs.

So, one question arises is that “can we mix cells of different heights to have a better tradeoff between performance, area and power, and to achieve a better design quality?”

**SLIDE: Motivation of Mixing Cell Heights**

We first compare the area and timing of designs synthesized with 12T-only, 8T-only and mixed cell heights in 28LP technology. Figures show the comparison. We can see that mixing cell heights can achieve more than 14% area reduction compared to 12T-only and 8T-only designs.

So mixing cell heights in a design is quite promising. But no existing flow can offer block-level mixed cell-height optimization.

Also, there is “chicken-and-egg” loop for such an optimization, where heights of cell rows are defined during floorplan stage before placement. But the choices of cell heights highly depend the on the placement solution.

**SLIDE: Also, There is Cost of Mixing Cell Heights**

Further, mixing cell heights is not free. “Interface area” must be inserted to align regions with different cell heights. So there is area cost.

As an example shown in the figure, the width of the interface area is four sites to align cell rows with different heights horizontally. And the height of the interface area is one M2 pitch to align cell rows with different heights vertically.

A mixed cell-height optimization must comprehend these cost and challenges.

**SLIDE: Overall Flow**

This slide shows our overall flow for mixed cell-height implementation.

We first synthesize design with libraries of all available cell heights.

Then, we perform placement of cells with modified cell LEF such that all cells have the same height. But we scale cell widths to maintain the same cell area. The figure shows one example. Such modification enables optimization with commercial P&R tools and resolves the “chicken-and-egg” loop between placement and cell height selection.

Then we partition the floorplan into regions with a specific cell height and legalize the placement solution so that each region contains only the cells with the same height. I will discuss the details in the following slides.

Afterwards, we revise the floorplan with determined cell heights from partitioning solution and insert interface area.

Then, we map cells back to their original heights and widths, and place them onto the updated cell rows.

Last, we perform the routing and routing optimization using commercial tools.

An example of the flow will be given in the next slide.

**SLIDE: Optimization Flow**

This slide shows an example of our optimization flow on design AES. In blue are 8T cells and in red are 12T cells. First figure shows the initial placement where with modified LEF, cells are “freely” placed with a commercial placer.

The second figure shows the partitioning solution based on dynamic programming. The yellow lines are cuts.

The third figure shows the legalized placement solution.

In the last two figures, we update the floorplan and map cells onto the updated cell rows. Note that in the last two figures, the floorplan contains both 12T cell rows and 8T cell rows.

**SLIDE: Floorplan Partitioning and Region Definition**

We partition die area into rectangular regions with different cell heights based on dynamic programming. To formulate the dynamic programming, we first define the cost of a region as the total area of minority cells within the region. As an example in the figure, the cost of a 12T region is the blue area within the region. We then sweep cut number (k), coordinates of a region, and all possible horizontal and vertical cuts to select the optimal solution with dynamic programming. The pseudo code is given here.

**SLIDE: Timing-Aware Placement Legalization**

Based on the partitioning solution, we legalize the placement with an iterative optimization. Our goal here is to have a legal placement such that a 12T region contains only 12T cells and an 8T region contains only 8T cells. We use two knobs in our iterative optimization. First one is cell displacement, as an example, we move a 12T cell from an 8T region to a 12T region. Second one is cell-height swapping, for example, we size a 12T cell in an 8T region to an 8T cell master. The figure shows our optimization framework. We build our optimizer in C++ and use TCL socket to interact with commercial P&R tool. To reduce the runtime of timing analysis, we also construct an internal timer based several delay models.

**SLIDE: Mapping Cells to Original Heights/Widths**

Recall that in the initial placement, we modify cell LEF such that all cells have the same height, and scale cells widths to maintain the same cell area. In the last stage, with the updated floorplan, we need to scale cells back to their original heights and widths and map them to updated cell rows.

We illustrate our flow with a special case – a 2D mesh. Based on a previous work on graph embedding, we can map cells to new cell rows in the updated floorplan with the maximum wirelength increase by (r + 1/r) times. Here r is the ratio between original cell height to the minimum cell height. An example of such graph embedding is shown in the figure. The initial graph is a 5x4 mesh, with wirelength = 1 for each edge. The vertical edges are not shown. Then we embed the graph into the right figure, which is a 4x5 mesh. In the new graph, the wirelength of a vertical edge is 1.25, the wirelength of a horizontal edge is 0.8.

**SLIDE: Benefits from Mixing Cell Heights**

The figure shows our experimental results at the post-routing stage. We compare the performance-area tradeoff of our mixed cell-height design with those of 8T- and 12T-only designs. We observe 25% area reduction compared to 12T-only case and 20% performance improvement compared to 8T-only case.

Our future works includes mixed cell-height clock tree synthesis and application of mixed cell-height optimization in 3DICs.

**SLIDE: Low-Cost Resilient Design Implementation**

Now, I will describe our work of the low-cost resilient design implementation.

**SLIDE: Background: Resilient Design**

First, what is a resilient design? A resilient design can detect and recover from timing errors. Therefore, it ensures correct operation with dynamic variations, such as supply voltage drop, temperature fluctuation, and cross-coupling. In addition, resilient design trades off between design robustness versus design quality. Last, by allowing timing errors, resilient design also improves performance through timing speculation. The bottom figure shows the energy comparison between conventional and resilient designs at different supply voltages. In conventional designs, timing violations lead to failure. Therefore, it is risky for voltage downscaling. Since resilient designs are able to detect and recover from timing errors, downscaling of supply voltage leads to power reduction. In this example, we see the energy gap can be 15%.

**SLIDE: Cost of Resilience is High**

Although there are significant benefits in resilient designs, the cost of resilience is high. As shown in the figures, resilient designs typically have additional circuits to detect timing errors, which inccur area and power penalties. Further, recovery from timig errors requires additional cycles, which causes throughput degradation. Also, error-tolerant flip-flops require large hold margin, which leads to short-path padding cost.

The table summarizes the cost of resilience for Razor, Razor-Lite and TIMBER. We can observe that the area and power penalties are quite high. Therefore, the goal in this work is to implement resilient designs such that their benefits overweigh their costs.

**SLIDE: Resilience Cost Reduction Problem**

We define the resilience cost reduction problem as – given an RTL design, throughput requirement, and error-tolerant registers, we implement design to minimize energy,

We estimate the design energy based on this equation. We further estimate throughput based on the equation shown in red dotted box, in which ER is the error rate; r is the number of cycles required to recover from an error; T is the clock period.

**SLIDE: Scope of This Work**

Most of previous works identify and optimize timing-critical and/or most frequently exercised paths instead of all paths to reduce resilience cost. But they ignore the tradeoff between resilience cost versus cost of datapath optimization. For example, when the number of Razor FFs reduces, the resilience cost reduces; but power and area of fanin circuits will increase.

This figure shows one example where we reduce the number of Razor flip-flops from 300 to zero. Red curve indicates the resilience cost, which includes power overhead of Razor flip-flops and throughput penalty. The blue curve indicates energy of non-resilient part of the design. And the green curve is the total energy. We see in this example that the total energy shows a bucket-shape curve due to tradeoffs between cost of resilience and cost of datapath optimization. In this work, we minimize the total energy using such tradeoff.

**SLIDE: Overview of Our Methodology**

This slide shows an overview of our methodology. Our flow starts with a pure-resilience solution, where all endpoints use error-tolerant flip-flops. Then we iteratively apply optimizations to reduce resilience cost.

We use two optimizations. First one is low-cost timing margin insertion. We call it as selective-endpoint optimization. In this optimization, we selectively increase margin at endpoints with timing violation, but at a low cost on datapath. Second, we use clock skew optimization to migrate timing slacks to endpoint with timing violation.

With the increased slacks, we are able to replace error-tolerant flip-flops to normal flip-flops and reduce the resilience cost.

**SLIDE: Overall Optimization Flow**

This slide shows the overall optimization flow. We iteratively optimize with selective-endpoint optimization and clock skew optimization. We start with an initial placement assuming that all endpoints use error-tolerant flip-flops. The right figure shows an example of slack distribution. We then insert timing margins on K paths based on sensitivity function. For enpoints which have larger slack than the required safety margin, we replace the error-tolerant flip-flops with normal flip-flops to reduce area and power penalties. We then apply activity-aware clock skew optimization to further optimize timing slacks. We keeps iterating between selective-endpoint optimization and clock skew optimization. After each iteration, we check whether the energy is less than the minimum energy ever seen. If so, we save the solution. The optimization flow stops when all endpoints are optimized.

In the following discussion I will give details on selective-endpoint optimization and clock skew optimization.

**SLIDE: Selective-Endpoint Optimization**

In the selective-endpoint optimization, we optimize the fanin cone with tighter timing constraints. Such optimization allows replacement of error-tolerant flip-flops with normal flip-flops. In other words, selective-endpoint optimization trades off cost of resilience versus cost of data-path optimization.

Two questions arise for the selective-endpoint optimization. First, which endpoint to be optimized? The second, how many endpoints should be optimized?

**SLIDE: Sensitivity Function**

Our answer to the first question is that we pick endpoints based on sensitivity functions. Our candidate sensitivity functions are listed at the left-hand side. To test the performance of each sensitivity function, we vary the number of endpoints to be optimized based on different sensitivity functions and compare the area and power penalties of their fanin cone. As shown in the figures, sensitivity function five incurs the minimum area and power penalty. So we use sensitivity function five in our optimization.

**SLIDE: Iterative Optimization**

For the second question, our optimization varies the number of optimized endpoints and we pick the minimum-energy solution.

In our selective-endpoint optimization, we pick top K endpoints which have the minimum sensitivity. Then we perform timing optimization on the fanin cone of these endpoints. If the optimized endpoints have positive slacks with respect to the safety margin, we replace them with normal flip-flops. We then perform error rate estimation. We check the design energy. If the energy is reduced, we store the current solution. W then update the sensitivity functions and continue with the iteration.

**SLIDE: Clock Skew Optimization**

Now I will discuss another optimization in our flow – the clock skew optimization. We use clock skew optimization to increas timing slacks on timing-critical and/or frequently-exercised paths. In the optimization, we first generate a sequential graph, in which each vertex is an endpoint and each edge is a timing path. The bottom figure shows one example. The weight of a path p-q is calculated based on this equation, in which slack p-q is the worst setup slcak of path p-q. Beta is a weighting factor; TG(p,q) is the toggle rate of path p-q.

Then we find the cycle of paths with the minimum total weight, and adjust clock latencies on the cycle to evenly distribute the weights. After the optimization, we contract the cycle into one vertex. We iterately execute Step 2 until all endpoints are optimized.

**SLIDE: Methodology Comparison**

This slide shows the optimization results with different methodologies. We compare our proposed methodology to two reference flows – the conventional methodology with only margin insertion; and a brute-force method which only use error-tolerant flip-flops for timing-critical paths.

As we can see in the results, our proposed flow achieves up to 21% energy reduction as compared to the reference flows. In addition, the small-, medium- and large-margin indicate cases with 1-sigma, 2-sigma and 3-sigma SS corners. We can see that using our proposed method, the resilience benefits increase with process variation.

**SLIDE: Energy Reduction from AVS**

We further perform the comparison at adaptive voltage scaling context. Since resilient design can detect timing errors, AVS allows a lower supply voltage for resilient designs to reduce power. Further, in the AVS context, our optimization is able to trade off timing-error penalty versus power reduction at a lower supply voltage.

Figures show the experimental results. We can see that with AVS, our proposed flow achieves an average of 17% energy reduction as compared to the pure-margin designs. The results also indicate that the resilience benefits increase in the context of AVS strategy.

*We implement resilient designs with a margin of 25% of clock period inserted on the paths that have normal FFs as endpoints. We then scale supply voltages on the implemented designs (as shown in the figures) to find the minimum-energy point, but without causing timing violation at endpoints with normal FFs at the slow corner.*

**SLIDE: Optimization of TIMBER-Based Designs**

We also apply our optimization to TIMBER-based designs. Since TIMBER flip-flops use time borrowing to mask timing errors, additional constraints have to be addressed for the optimization. First, there must be no loop of TIMBER flip-flops. Second, there cannot be chained TIMBER flip-flops with more than two stages. Also, time borrowing requires additional timing slacks on fanout paths of TIMBER flip-flops.

The figure shows our experimental result on ARM Cortex M0 at foundry 40nm technology. We assume an error detection interval of 10% of the clock period in this example. We observe that our optimization flow achieves 23% and 7% energy reduction compared to the pure-margin design and the brute-force method, respectively.

**SLIDE: Optimization for Mix-and-Match Die Stacking in 3DICs**

Now I will describe our design-stage optimization for mix-and-match die stacking in 3DICs.

**SLIDE: Mix-and-Match Die Stacking**

First, what is mix-and-match die stacking? It is an interesting idea proposed by previous works to stack slow dies with fast dies to improve the parametric yield in 3DICs.

The figure shows one example. The numbers indicates die speed information measured before 3D integration. Based on the information, Ferri08 integrates slow CPU dies with fast L2 cache dies, and fast CPU dies with slow L2 cache dies.

**SLIDE: Design Stage Optimization for Mix-and-Match**

This figure shows the timing slack improvement due to mix-and-match stacking. In the example, SS-SS indicates slow Tier 0 + slow Tier 1, SS-FF indicates slow Tier 0 + fast Tier 1. We can see that the mix-and-match stacking offers 75ps timing improvement over the conventional worst-case analysis.

However, there is no holistic design for eventual stacking of 3DICs. In other words, no work considers the mix-and-match stacking during the design stage.

The figure shows a simple example of how partitioning solution affects design signoff timing in the regime of mix-and-match stacking. In the example, with mix-and-match stacking, blue cut leads to the maximum timing slack, while red cut leads to the minimum timing slack.

Therefore, our goal in this work is to study design-stage optimization for mix-and-match stacking.

**SLIDE: Challenges in Mix-and-Match-Aware Partitioning**

However, mix-and-match-aware partitioning is not trivial.

First, the optimal cut location on one path can conflict with those on the other paths. The right figure shows one example that has different optimal partitioning solutions for different objectives. The first one optimizes the slack on path A-C. The second one optimizes the slack on path B-C. The third one optimizes the worst slack over two paths. And the fourth one optimizes the worst slack over two paths with large vertical interconnect delay impact.

Second, partitioning is not free. There is delay impact of vertical interconnect insertion.

Last, process variation can lead to asymmetric path delay distribution, which increases the difficulty for mix-and-match-aware partitioning.

In the following slides, I will describe our proposed methodologies for mix-and-match-aware partitioning.

**SLIDE: ILP-Based Partitioning Method**

We first propose an ILP-based partitioning method. The slide shows the formulation and notations. In the formulation, we minimize the maximum path delay.

These two constraints force beta to be one if there is a cut between cell i and cell i’.

This constraint calculate the path delay in the regime of mix-and-match stacking. dj and dj’ respectively indicate cell delay at process corner j and process corner j’. dVI is the delay impact of VI insertion. Last two constraints specify the area-balancing criterion.

Although the ILP-based method achieves near-optimal solution, it has large runtime. So we also propose a heuristic partitioning method.

**SLIDE: Heuristic Partitioning Method (1)**

Our heuristic method has two steps. The first step performs maximum-cut partitioning on timing-critical sequential graph.

We first classify the paths into three categories according to their slacks and the vertical interconnect delay impact.

Type-I is the timing non-critical paths, which we do not optimize.

Type-II is the timing-critical path without tolerance of VI insertion, of which the impact of VI insertion is larger than the timing benefits from mix-and-match stacking. We want to force the startpoint and endpoint of such paths to be on the same tier.

Type-III is the timing-critical paths with tolerance of VI insertion. We want to force the startpoint and endpoint of such paths to be on different tiers.

In step 2, we extract the restricted sequential graph that contains only Type-II and Type-III paths.

We then collapse vertices connected with Type-II paths into one vertex.

On the updated graph, we perform maximum cut to partition the flip-flops into two tiers.

**SLIDE: Heuristic Partitioning Method (2)**

In the second step of our heuristic we perform timing-aware multi-phase FM partitioning.

One issue for the FM partitioning is that it is hard to foresee slack benefits with the existence of VI delay impact.

As shown in the figure, moving one cell across tiers degrades the slack by 70ps due to VI delay impact. But the following moves on its neighbor cells compensate the VI delay impact and achieve 50ps timing improvement.

To resolve such issue, we propose to cluster cells with a given range of cluster size, so that the timing benefits can compensate the VI delay impact.

The bottom figure shows our optimization flow. Based on an initial partitioning solution, we perform multi-thread optimization. Different threads cluster cells with different cluster sizes, followed by timing-aware FM optimizations. Then we pick the solution with the maximum slack as the input to the next phase.

**SLIDE: Calibration of Heuristic Partitioning**

We calibrate our heuristic partitioning method by compare its solution to that of ILP-based method. We perform such comparison with different VI delay impact, process corners. We observe that the heuristic method leads to only less than 30ps slack different compared to the ILP-based partitioning solution.

**SLIDE: Validation of Our Method**

We also extend two existing 3DIC implementation flow with our partitioning method for mix-and-match. We observe from the results that our optimization achieves up to 16% performance improvement in the regime of mix-and-match compared to the existing flows.

One of our future works is to combine stacking optimization and design-stage optimization for mix-and-match stacking.

**SLIDE: Roadmap of My Thesis**

This slide shows the overview of the roadmap of my thesis. I am planning to have three parts – the mixed-fabric optimization, low-power optimization and multi-corner optimization. The current and target scopes are shown.

**SLIDE: Thank You!**

Thank you again.