NOLO : A No-Loop, Predictive Useful Skew Methodology for Improved Timing in IC Implementation

Tuck-Boon Chan[‡], Andrew B. Kahng[†‡] and Jiajia Li[‡]
[†]CSE and [‡]ECE Departments, UC San Diego, La Jolla, CA 92093
{tbchan, abk, jil150}@ucsd.edu

***Abstract*—Useful skew is a well-known design technique that adjusts clock sink latencies to improve performance and/or robustness of high-performance IC designs. Current design methodologies apply useful skew after the netlist has been synthesized (e.g., with a uniform skew or clock uncertainty assumption on all flops), and after placement has been performed. However, the useful skew optimization is constrained by the zero-skew assumptions that are baked into previous implementation steps. Previous work of Wang et al. [15] proposes to break this chicken-egg quandary by back-annotating post-placement useful skews to a re-synthesis step (and, this loop can be repeated several times). However, it is practically infeasible to make multiple iterations through re-synthesis and physical implementation, as even the time for placement alone of a large hard macro block in a 28nm SOC can be five days [10]. Thus, in our work we seek a *predictive, one-pass* means of addressing the chicken-egg problem for useful skew.**

We observe that in a typical chip implementation flow, timing slacks at post-synthesis stage do not correlate well with timing slacks at post-routing stage. However, the correlation is improved when useful skew is applied at the post-synthesis stage. Based on this observation, we propose NOLO, a simple, "no-loop" predictive useful skew flow that applies useful skew at post-synthesis within a one-pass chip implementation. Further, our predictive useful skew flow can exploit an additional synthesis run to improve circuit timing without any turnaround time impact (two synthesis steps are run in parallel). Experimental results in a 28nm FDSOI technology show that our predictive useful flow can reduce runtime by 66% and improve total negative slack by 5% compared to the useful skew back-annotation flow of [15].

I. Introduction

Zero-skew clock tree synthesis is commonly used in conventional chip implementation flows to minimize the maximum clock skew. Figure 1 shows a conventional chip implementation flow, in which we synthesize a design described in RTL to obtain a gate-level netlist. We then place the gate-level netlist, perform clock tree synthesis (CTS) based on the placement results, and route the connections in the design. We refer to this as a *zero-skew flow*.

By intentionally skewing clock latencies[1] of flip-flops (*flops*), we can increase the timing slacks on critical paths while still satisfying the timing constraints on non-critical paths [6][11]. This skew scheduling methodology for timing optimization is well-known as *useful skew*. Previous works that study useful skew mainly focus on two objectives – (i) to minimize the clock period and (ii) to maximize the timing margin (robustness). Fishburn [6] formulates a linear program (LP) to optimize clock latencies for performance improvement. The LP formulation considers both setup and hold constraints. Szymanski [11] further improves the efficiency of the LP by selectively generating constraints. Wang et al. [12] also propose an LP-based approach to evaluate potential slacks in circuits and optimize clock skew. The clock skew optimization problem can also be solved by graph-based methods as in [5].

More recent work of Albrecht et al. [1] [2] formulates useful skew optimization as a *maximum mean weight cycle* (MMWC) problem, which optimizes not only the minimum slack in a circuit, but also the slacks on other paths. The MMWC approach achieves better timing improvement than the LP-based approach, and is currently the standard approach for useful skew optimization in commercial EDA tools. Runtimes are reduced using faster MMWC algorithms such as [16][17].

Figure 2 shows a *typical useful skew flow*, in which the clock latencies are optimized after synthesis, placement and CTS in the *Skew_opt* step. A crucial observation is that the typical useful skew flow suffers from a "chicken-and-egg" quandary: after the netlist has been synthesized and placed with zero skew, what useful skew can accomplish is limited.

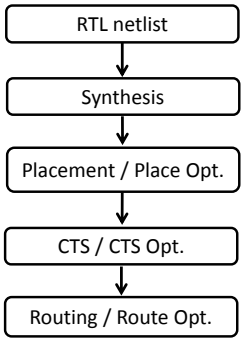


Fig. 1: A conventional zero-skew chip implementation flow (*zero-skew flow*).

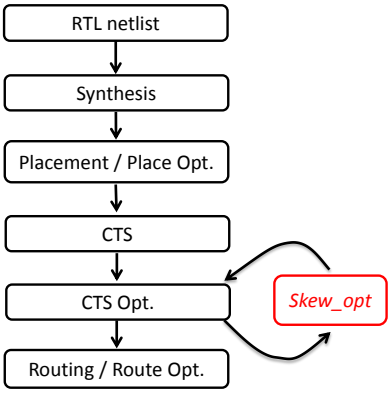


Fig. 2: A standard useful skew flow (*typical useful skew flow*).

To fully exploit the potential of useful skew, Albrecht et al. [3] interleave useful skew with RTL synthesis to optimize the performance and area of a design. Hurst et al. [9] propose a placement algorithm with a tight integration of useful skew to minimize maximum mean delay in any circuit loop. Although these methods can inject useful skew into synthesis or placement stages of implementation, substantial changes would be required to implement them with existing commercial tools. Thus, the work of Wang et al. [15] is notable for its feasibility with modern back-end EDA tools: the authors propose to back-annotate post-placement clock latencies (obtained from useful skew optimization) to the pre-synthesis stage, and re-execute the flow. I.e., after feeding back the clock latencies, [15] re-performs synthesis and placement, followed by another useful skew optimization (see Figure 3). This synthesis, placement and useful-skew loop continues until there are no further improvements; empirical results in [15] imply that only two iterations are required to realize the benefits of the proposed methodology.

[1]We define clock latency as the delay from the clock source to a flip-flop clock input pin.

Our Work

Although the back-annotation flow can account for interactions between synthesis, placement and useful skew optimizations, having such a loop in the flow has unacceptable turnaround time impacts. According to [10], it is practically infeasible to make multiple iterations through re-synthesis and physical implementation, as even the time for placement alone of a large hard macro block in a 28nm SOC can be five days (and, a single pass through placement + placeOpt + CTS can have over a week of runtime). This motivates us to seek a *predictive, one-pass* means of addressing the chicken-egg problem for useful skew.

To avoid turnaround time impact, we *predict* and enforce useful skews at the post-synthesis stage, within a one-pass implementation. As outlined in Figure 4, our new NOLO ("no-loop") flow predicts useful skews based on timing analysis of the synthesized netlist using the default wireload model provided in timing libraries. Experimental results in Section IV show that our simple prediction flow achieves good timing quality compared to a **Typical** useful skew flow without only a single implementation pass (i.e., no runtime penalty). We further improve circuit timing with a variant flow (the dotted box in Figure 4) that predicts the useful skews based on *two* synthesized netlists. With the optional flow, we can improve total negative slack by 5% compared to the back-annotation flow of [15]. Note that the additional synthesis run has no turnaround impact as we can launch both synthesis runs in parallel.

To complete our study, we also implement a wide range of alternative back-annotation flows (e.g., post-routing information can be fed back to synthesis, to placement, or to clock tree synthesis stages) to experimentally assess their runtime and timing quality tradeoffs.

Our discussion below will use the following definitions.

- *Zero-skew flow :* the conventional chip implementation flow with zero-skew CTS.
- *Typical useful skew flow :* one-pass chip implementation flow with useful skew optimization using a commercial tool, e.g., *skew_opt* in *Synopsys IC Compiler* [21].
- *Back-annotation flow :* a chip-implementation flow that feeds back circuit information to earlier stages for useful skew optimization. Variants of back-annotation flows are described in Section III.
- *Prediction flow :* **our** new one-pass chip implementation flow, NOLO, with useful skew optimization at the post-synthesis stage.

We use *slack* to denote the endpoint setup slack on maximum-delay paths between sequentially adjacent flops or ports [7]. Furthermore, since MMWC is the de facto standard approach for useful skew optimization, we perform useful skew scheduling using the *maximum mean weight cycle* formulation of [2] and the algorithms given in [1]. Thus, (i) our useful skew optimization is same as that in the back-annotation flow of Wang et al. [15], and (ii) we assume that the "typical useful skew flow" also optimizes the skew schedule using the MMWC formulation.

Scope and Organization of Paper

Our work achieves the somewhat surprising result that an improved useful skew optimization at the post-synthesis stage can enable a single-pass flow to achieve similar or better timing improvements compared to back-annotation flows. We focus on optimization of useful skews rather than the downstream physical implementation (i.e., CTS, placement and routing with given useful skews). Our three main contributions are summarized as follows.

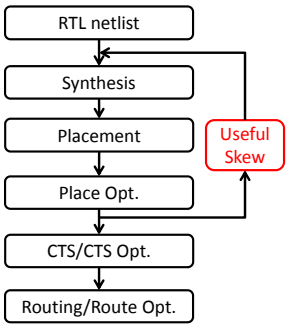


Fig. 3: A chip implementation flow with useful skew back-annotation (*back-annotation flow*).

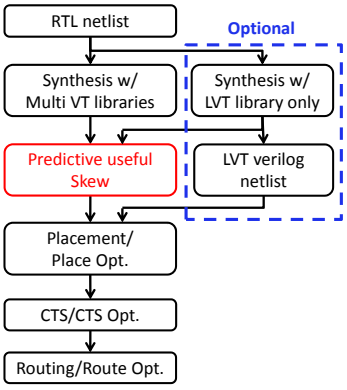


Fig. 4: Our predictive NOLO ("no-loop") useful skew flow (*prediction flow*).

1) We show that applying useful skews at post-synthesis stage of circuit implementation improves the timing correlation between post-synthesis stage and post-routing stage.
2) We also show that with an additional synthesis run, our predictive useful skew flow can achieve better timing slacks compared to back-annotation flows.
3) We implement different useful skew flows to study the tradeoffs between runtime and timing slacks (with the same area and power).

We present our NOLO prediction flow in Section II. Section III describes our experimental setup and implementation details of different useful skew flows. We report experimental results in Section IV. Section V concludes our discussion and gives several directions for future work.

II. Predictive Useful Skew Methodology

Our predictive flow applies useful skew optimization to a post-synthesis netlist, such that the useful skew optimization is not affected by an initial placement, and allows for a one-pass chip implementation flow.

A. Analysis of the Impact of Placement and Timing Optimization

Intuitively, applying predicted useful skews at the post-synthesis stage is risky, in that timing information at this stage is incomplete. In other words, the circuit timing will be changed by subsequent placement, routing and optimization steps (e.g., cell resizing and/or swapping, buffer insertion, cloning, parasitics from wiring, etc.). To gain initial understanding of the impact of a predictive useful skew flow at the post-synthesis stage, we run two basic implementation flows as illustrated in Figure 5.

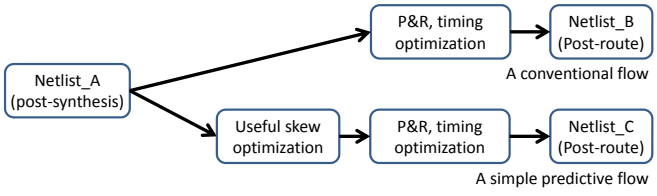


Fig. 5: Overview of two basic implementation flows.

Given a post-synthesis netlist (*Netlist_A*), we run placement and routing (P&R) to obtain a post-routing netlist without any useful skew

optimization (*Netlist_B*). Meanwhile, we extract timing information from *Netlist_A*, and apply MMWC-based useful skew optimization. Based on the useful skew results, we annotate clock latencies in an *.sdc* file and run the same P&R flow to obtain another post-routing netlist (*Netlist_C*).

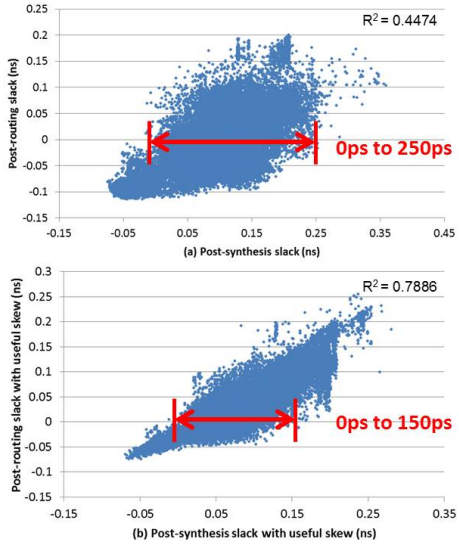


Fig. 6: Timing slacks at post-synthesis versus timing slacks at post-routing stage: (a) without useful skew, and (b) with useful skew. Paths are extracted from the *mpeg2* testcase with 0.4ns clock period (Table I).

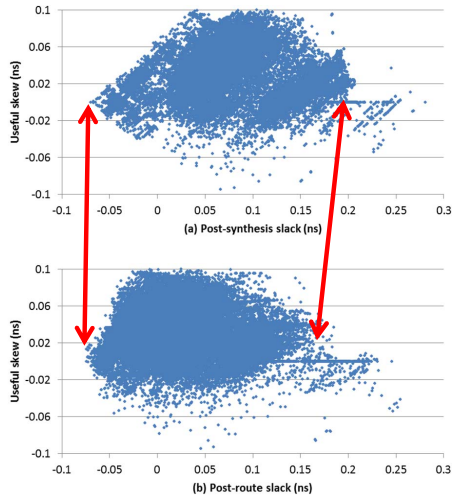


Fig. 7: Useful skew versus timing slacks at (a) post-synthesis and (b) post-routing stages. Paths are extracted from the *mpeg2* testcase with 0.4ns clock period (Table I).

Figure 6 shows the timing slacks (for all sequentially adjacent flop pairs) at post-synthesis stage versus the timing slacks at the post-routing stage. In Figure 6(a), we can see that in a chip implementation flow without any useful skew optimization (i.e., the top flow in Figure 5), the timing slacks at post-synthesis stage have poor correlation with the timing slacks at post-routing stage. For example, critical paths at post-routing stage (timing slack = 0) correspond to the paths with 0ps to 250ps timing slacks at post-synthesis stage. On the other hand, Figure 6(b) shows that with useful skew optimization at post-synthesis stage, the timing slacks at post-synthesis and post-routing stages have much better correlation. More specifically, the critical paths at post-routing stage (timing slack = 0) correspond to the paths with 0ps to 150ps timing slacks at post-synthesis stage when useful skew is applied at post-synthesis stage. This is because the useful skew optimization at post-synthesis relaxes the timing constraints. As a result, the P&R stages do not need to significantly perturb the netlist to meet the timing constraints. Further, Figure 7 shows that the relative values of useful skew and timing slacks are similar for post-synthesis and post-routing stages. The post-routing slack is slightly smaller due to the impact of interconnect delay and power/area optimization during the P&R stage.

A Key Observation. Because of the good correlation between timing slacks at post-synthesis stage and post-routing stages, the clock latencies resulting from useful skew optimization are similar at these two stages. Therefore, we expect that applying useful skew optimization at post-synthesis stage will lead to similar timing improvements compared to applying useful skew optimization at later stages. We validate this hypothesis by generating the optimal useful skews at post-routing stage (*Netlist_C*) and comparing with the predictive useful skews generated at post-synthesis stage (*Netlist_A*). Each dot in Figure 8 represents the useful skew of a pair of sequentially adjacent flops. The x-axis is the optimal useful skew at post-synthesis stage and the y-axis is the optimal useful skew at post-routing stage; the correlation coefficient for the useful skews is 0.83. Since the predicted useful skews at the post-synthesis stage are very similar to the **optimal** useful skews at the post-routing stage, our predictive useful skew flow would seem likely to achieve near-optimal timing quality. Note that the results in Figures 6 to 8 are representative for all other testcases in our study.

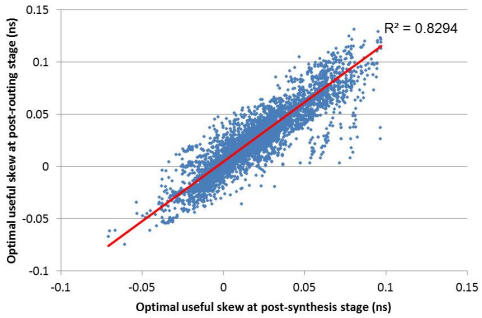


Fig. 8: Optimal useful skews (obtained from MMWC) based on timing information at post-synthesis and post-routing stages have good correlation. Paths are extracted from the *mpeg2* testcase with 0.4ns clock period (Table I). This suggests why simple prediction of useful skews at the post-synthesis stage is feasible.

B. Implementation of Predictive Useful Skew Flow

It is well known that useful skew optimization migrates timing slack from a non-critical path to the sequentially adjacent critical paths. Thus, the maximum achievable timing slack is bounded by the mean timing slack of paths that form a cycle. Therefore, we follow standard practice and formulate the useful skew optimization as the *maximum mean weight cycle* (MMWC) problem [1] [2]. Given a post-synthesis netlist with edge-triggered flops, we model the netlist using the directed graph $G(V,E)$, where each flop in the

netlist is represented by a vertex[2] and there is an edge between two vertices whenever there is a purely combinational path between the corresponding flops. The setup and hold slacks on the path are modeled by the following equations

$$s_{i,j,setup} = -x_i + x_j + T - d_i - d_{i,j}^{max} - t_j^{setup}$$
$$s_{i,j,hold} = x_i - x_j + d_i - d_j + d_{i,j}^{min} - t_j^{hold}$$

where $s_{i,j,setup}$ and $s_{i,j,hold}$ are respectively the setup and hold slack on the path from the i^{th} flop (f_i) to the j^{th} flop (f_j). x_i denotes the clock latency of the i^{th} flop. T is the clock period, d_i is the clock-to-Q delay of f_i, and $d_{i,j}^{max}$ and $d_{i,j}^{min}$ are respectively the maximum and minimum path delay from f_i to f_j. Last, t_j^{setup} and t_j^{hold} are the setup and hold time of f_j, respectively. We then formulate our useful skew optimization as

$$\begin{aligned} \text{maximize} \quad & \sum_{i,j} s_{i,j,setup} \\ \text{s. t.} \quad & s_{i,j,hold} \geq 0, \forall i,j \end{aligned} \tag{1}$$

We optimize the sum of setup slacks (flop pairs) because a larger setup slack can potentially improve the achievable operating frequency, or be traded off for power and area recovery. We also consider hold time constraints to ensure correct circuit operation. In the MMWC optimization, we first calculate the weight of each edge (i.e., the worst setup slack corresponding to a pair of flops). We then find the minimum-weight edge in each iteration and label it as a critical path. For an efficient implementation, we determine the minimum-weight edges using the *parametric shortest path* algorithm (details of which are given in [1]). When the critical paths form a cycle, we set the weight (i.e., timing slack) of each edge on the cycle as the maximum mean weight of the cycle. Based on the timing slack, we then determine the clock latency for each vertex (flop). After assigning the clock latencies, we contract the cycle into one vertex and update the weights of incoming/outgoing edges of the contracted vertex. We iteratively search for the minimum-mean-weight cycle and contract the cycle until every vertex is assigned to a clock latency. To incorporate hold constraints in the MMWC, we add edges in parallel to the edges corresponding to setup slack (but with reversed direction). Similarly, each of these (hold) edges is given a weight that corresponds to the hold slack. The parametric shortest path algorithm will honor the constraints defined by hold edges when it searches for the minimum (setup) weight edge.

C. An Improved Predictive Useful Skew

The solution quality of useful skew optimization at the post-synthesis stage will be affected by various timing optimizations during place and route, such as VT-swapping and sizing. To address this issue, we also predict useful skews based on a netlist synthesized with only the fastest available cells (e.g., low threshold voltage (LVT) library) (Algorithm 1). Prediction of useful skews based on the LVT-only netlist not only comprehends the impact of VT-swapping in later-stage optimizations, but also estimates the achievable slack between each flop pair. However, hold time analysis on a netlist with only the fastest cells is too conservative. Thus, we also propose to synthesize the design with multiple libraries (e.g., multi-VT cell libraries) and formulate the hold constraints based on the multi-VT netlist (Line 4 in Algorithm 1). As shown in Algorithm 1, this prediction flow requires two synthesis runs, which can be executed in parallel so that there is no turnaround time impact. Based on the synthesized LVT and multi-VT netlists, we optimize useful skews using the MMWC algorithm (Line 5). We then use the LVT netlist for placement and routing (P&R) (Line 6). Note that we use multi-VT libraries for P&R implementations, i.e., the P&R tools will optimize power by swapping LVT cells to other VT flavors on *non-critical timing paths*. Thus, the accuracy of our useful skew prediction based on LVT-only netlist is less affected by the VT swapping. In the following discussion, we use **SimPred** to refer to the simple prediction flow described in Section II-B, and **ImpPred** for this improved predictive useful skew flow based on two synthesis runs.

Algorithm 1 No-loop, Predictive Useful Skew Methodology

Procedure *ImpPred*(RTL, *.sdc*, $Liberty_{LVT}$, $Liberty_{MVT}$)
Output: N_{out}
1: $N_{LVT} \leftarrow$ Synthesis(RTL, *.sdc*, $Liberty_{LVT}$);
2: $N_{MVT} \leftarrow$ Synthesis(RTL, *.sdc*, $Liberty_{MVT}$);
3: $V \leftarrow$ flops, PIs, POs in N_{LVT};
4: $E \leftarrow$ max-delay paths in $N_{LVT} \cup$ min-delay paths in N_{LVT};
5: clock latencies $\leftarrow$ MMWC(V, E);
6: $N_{out} \leftarrow$ P&R(N_{LVT}, *.sdc*, $Liberty_{LVT}$, clock latencies);

III. Experiment Setup

Our experiments use a dual-VT 28nm FDSOI library and three RTL designs from the *OpenCores* website [19]. We show statistics of testcases (including clock period, total number of cells, number of flops, and number of maximum/minimum delay paths (i.e., number of edges in the sequential graph)) in Table I. We use *Synopsys Design Compiler vH-2013.03-SP3* [20] to synthesize the RTL netlists.[3] We run P&R using *Synopsys IC Compiler vH-2013.03-ICC-SP3* [21]. We also use *Synopsys IC Compiler* for power analysis, and *Synopsys PrimeTime H-2013.06-SP2* [22] for timing analysis. The setups for timing analysis are given in Table II, where in the absence of AOCV tables we use timing derates to model on-chip variation. All (dual-VT) implementation experiments are run with two signoff corners at {125°C, 0.9V, SS} and {-40°C, 1.05V, FF}. To mitigate the effects of tool noise [8], each P&R implementation executes three separate runs with small perturbations of clock period (i.e., -1ps, 0ps, +1ps); we report the largest endpoint slack results obtained over all three final-routed netlists.

TABLE I: Benchmark designs

Design	Clk period (ns)	#Cells	#Flip-flops (#Vertices)	#Paths (#Edges)
aes_cipher	0.6	∼23k	530	16251
des_perf	0.5	∼11k	1985	23153
jpeg_encoder	0.6	∼50k	4712	137333
mpeg2	0.4	∼11k	3381	95490

The back-annotation flow can have different variants. In addition to the back-annotation flow proposed in [15], we have implemented four variant back-annotation flows, designated as **BA-I**, **BA-II**, **BA-III** and **BA-IV**.

In **BA-I** (Figure 9), we collect timing information at post-placement stage, optimize useful skew, and back-annotate the clock

[2]Following guidance from [4], all input (resp. output) ports are merged and treated as a single vertex in our MMWC useful skew optimization. This step enables every maximum-delay combinational path (flop-flop, PI-flop or flop-PO) to be included in at least one cycle.

[3]A physical synthesis flow is used: We first run the default synthesis flow, then implement a fast placement of the synthesized netlist, based on which another pass of synthesis is made with topographical ("topo") option.

TABLE II: Experimental setups for timing analysis

Parameter	Value
Clock uncertainty (synthesis)	0.15 × clock period
Clock uncertainty (placement, CTS)	0.10 × clock period
Clock uncertainty (CTS opt, routing)	0.05 × clock period
Maximum transition	0.08 × clock period
Timing derate on net delay (early/late)	0.90 / 1.19
Timing derate on cell delay (early/late)	0.90 / 1.05
Timing derate on cell check (early/late)	1.10 / 1.10

latencies to the post-synthesis stage. For **BA-II**, **BA-III** and **BA-IV**, we collect timing information at post-routing stage and optimize useful skew. The optimized clock latencies are then back-annotated to the synthesis, placement, and CTS stages, respectively, in **BA-II**, **BA-III** and **BA-IV**.

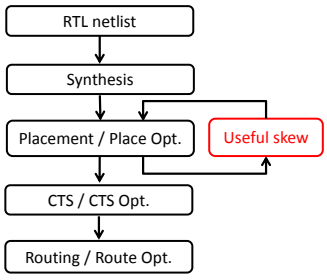


Fig. 9: BA-I flow.

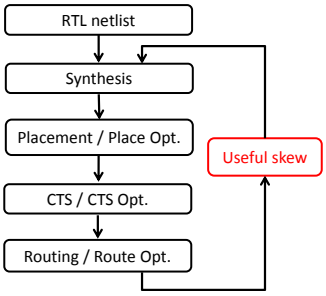


Fig. 10: BA-II flow.

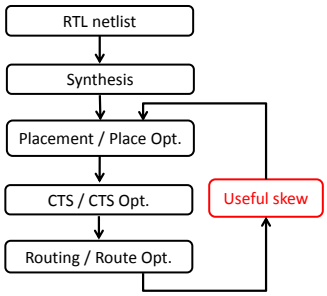


Fig. 11: BA-III flow.

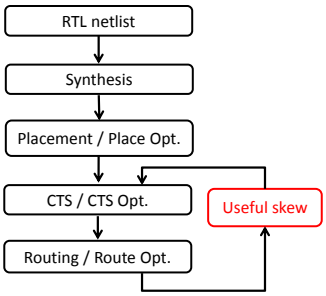


Fig. 12: BA-IV flow.

IV. Experimental Results

We perform chip implementations on designs (listed in Table I) with eight chip implementation flows – (a) the standard useful skew flow (**Typical**) where we use the command *skew_opt* in *Synopsys IC Compiler* [21] to generate desired clock latencies for incremental clock tree optimization; (b) the back-annotation flow by Wang et al. [15] (**BA-W**), which is depicted in Figure 3; (c) four variants of back-annotation flows described in Section III; and (d) our two NOLO ("no-loop") predictive flows (**SimPred** and **ImpPred**), in which we apply predicted useful skews at post-synthesis stage and continue to use them throughout timing optimization in P&R.

Results in Table III show that different flows achieve similar power and area. Also, all designs are free of any hold time violation. Thus, we achieve clean comparisons of different flows based on the total negative slack.

Back-annotation vs. Typical: Results in Table III show that the **BA-W** flow can achieve better total negative slack (TNS) compared to the **Typical** flow (average across all testcases in Table I). This is mainly because the useful skew optimization in the **BA-W** flow can interact with the synthesis and placement stages through the feedback loop. As a result, the cells on critical paths can be re-sized, re-structured and/or re-allocated to improve timing quality. However, the runtime of the **BA-W** flow is 85% longer than the **Typical** flow.

SimPred vs. Back-annotation: Results in Table III show that although the **SimPred** flow can also achieve significant improvement compared to the **Typical** flow, the average TNS achieved by the **SimPred** flow is approximately 20% worse compared to **BA-W**. This is expected because the useful skew solution (at post-synthesis stage) may be suboptimal due to design changes in the place and route stages. However, the **SimPred** reduces runtime by 66% compared to the **BA-W** flow.

ImpPred vs. Back-annotation: Our results also show that with the concurrent LVT-only synthesis run, the **ImpPred** flow achieves improved TNS, power and area (on average) compared to **BA-W**. This is because the benefit of useful skew optimization is limited by the zero-skew placement in **BA-W**. For example, buffers are inserted in the zero-skew netlist to fix timing violations, which increases area and power. Moreover, the critical paths will not fully exploit the potential benefits of useful skew. In contrast, our **ImpPred** flow relaxes timing constraints at the post-synthesis stage via an early-stage useful skew optimization (see Section II-C). We believe that this enables the optimized netlist to meet timing constraints with less area and power penalty (e.g., less buffer insertions).

Among the four testcases, **BA-W** only does better for the *jpeg_encoder* testcase, by a small margin. Overall, our prediction of useful skew at post-synthesis stage is superior to the **BA-W** back-annotation flow. Moreover, our **ImpPred** is a **one-pass** implementation which reduces runtime by 66% compared to **BA-W**. Note that the runtime of the **ImpPred** flow is smaller than the runtime of the **SimPred** flow, even though **ImpPred** implements two synthesis runs. This is because we execute the synthesis runs simultaneously, and the improved timing quality leads to a faster convergence in the P&R stages.

Design Dependencies: We observe that the improvements from useful skew implementations are design-dependent. Timing improvements with useful skew are less for a design with fewer flops, because the number of paths that can be improved is smaller (e.g., *aes_cipher*). In this work, we have focused only on optimization of timing. Conventional wisdom would suggest that our improvements in timing can be traded for power and area improvements, and we plan to consider the tradeoffs between timing and power/area objectives in our future work.

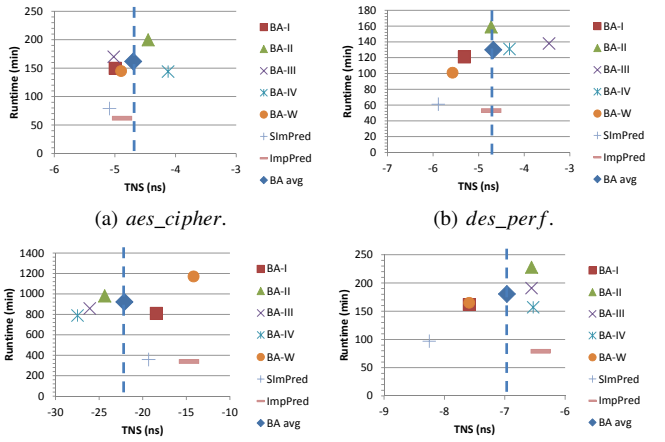
Comparison Among Variants of Useful Skew Flows: We compare the runtime and resultant total negative slacks of various useful skew flows. In the back-annotation flows, we iteratively optimize until the improvement in the average setup slack is less than 50ps. All the back-annotation flows converge within three iterations.

Figure 13 shows that the TNS values of the back-annotation flows vary depending on the testcase. This suggests that even with back-annotation, the useful skew optimization may be misled by the initial netlist and thus end up with suboptimal solutions. Since the back-annotation flows achieve different TNS values, we also plot the average TNS of all back-annotation flows (including **BA-W**) for comparison (i.e., the blue diamond symbol and dotted lines). The results show that **ImpPred** can achieve better results compared to the average TNS of the back-annotation flows (*BA avg*) for larger testcases (*jpeg_encoder* and *mpeg2*). For smaller testcases (*aes_cipher* and *des_perf*), **ImpPred** achieves similar TNS

TABLE III: Design metrics of routed design from different flows.

Design	Flow	Power (mW)	Area (μm)	#Hold vio.	TNS (ns)	WNS (ns)	Runtime (min)
aes_cipher	Typical	16984	35.8	0	-7.806	-0.047	117
	BA-W	16860	35.0	0	-4.898	-0.042	145
	SimPred	16539	34.7	0	-5.089	-0.035	79
	ImpPred	16002	34.3	0	-4.883	-0.036	62
des_perf	Typical	21971	65.8	0	-13.920	-0.046	108
	BA-W	20445	61.2	0	-5.574	-0.032	101
	SimPred	20603	62.2	0	-5.885	-0.034	61
	ImpPred	19618	57.2	0	-4.726	-0.035	53
jpeg_encoder	Typical	72799	77.0	0	-136.650	-0.131	496
	BA-W	58874	64.6	0	-14.166	-0.043	1171
	SimPred	57878	63.4	0	-19.317	-0.043	358
	ImpPred	56970	61.7	0	-14.695	-0.045	339
mpeg2	Typical	27655	52.6	0	-137.855	-0.168	134
	BA-W	25761	48.5	0	-7.590	-0.049	165
	SimPred	25415	48.3	0	-8.251	-0.054	97
	ImpPred	25250	48.4	0	-6.408	-0.046	79
Average of 4 designs	Typical	34852	57.8	0	-74.058	-0.098	213
	BA-W	30485	52.3	0	-8.057	-0.042	395
	SimPred	30108	52.2	0	-9.636	-0.042	148
	ImpPred	29460	50.4	0	-7.678	-0.041	133

compared to the average of back-annotation flows. Also, it is clear that our predictive flows have significantly less runtime than the back-annotation flows for all testcases.



(a) *aes_cipher*. (b) *des_perf*. (c) *jpeg_encoder*. (d) *mpeg2*.

Fig. 13: Comparison among useful skew flows. Our **ImpPred** flow achieves better or similar TNS but with 66% runtime reduction compared to back-annotation flows.

V. Conclusions

We propose NOLO, a "no-loop" predictive useful skew optimization flow, based on timing information of a post-synthesis netlist. To account for the potential of timing changes during the place and route stages, we improve our estimate of potential slack in the netlist by running an additional logic synthesis step using fast library cells. Based on this technique, we show that an improved predictive useful skew flow (**ImpPred**) can achieve similar or better total negative slack compared to back-annotation flows, with only one pass through chip implementation. The runtime of our predictive useful skew flows is similar to the runtime of the **Typical** flow, which is approximately 66% less than the runtime of the back-annotation flow in [15].

Our study of different back-annotation flows indicates that back-annotation (or optimization loops) cannot completely resolve the "chicken-and-egg" problem. We see that the timing quality varies depending on testcases. This is because even with back-annotation, the useful flows can be misled to a suboptimal local solution.

There are two major directions for our future work. First, we plan to analyze and apply our useful skew flows across multiple PVT corners. Second, we plan to study and develop models of the tradeoffs among area, power and timing with useful skew.

References

[1] C. Albrecht, "Efficient Incremental Clock Latency Scheduling for Large Circuits", *Proc. Design Automation and Test in Europe*, 2006, pp. 6-10.

[2] C. Albrecht, B. Korte, J. Schietke and J. Vygen, "Maximum Mean Weight Cycle in a Digraph and Minimizing Cycle Time of a Logic Chip", *Discrete Applied Mathematics* 123(1-3) (2002), pp. 103-127.

[3] C. Albrecht, P. Witte, and A. Kuehlmann, "Performance and Area Optimization using Sequential Flexibility", *Proc. International Workshop on Logic and Synthesis*, 2004.

[4] C. Albrecht, *personal communication*, July 2013.

[5] R. B. Deokar and S. S. Sapatnekar, "A Graph-theoretic Approach to Clock Skew Optimization", *Proc. International Symposium on Circuits and Systems*, 1994, pp. 407-410.

[6] J. P. Fishburn, "Clock Skew Optimization", *IEEE Transactions on Computers* 39(7) (1990), pp. 945-951.

[7] E. G. Friedman, *Clock Distribution Networks in VLSI Circuits and Systems*, New York, IEEE Press, 1995.

[8] K. Jeong and A. B. Kahng, "Methodology From Chaos in IC Implementation", *Proc. International Symposium on Quality Electronic Design*, 2010, pp. 885-892.

[9] A. Hurst, P. Chong and A. Kuehlmann, "Physical Placement Driven by Sequential Timing Analysis", *Proc. International Conference on Computer-Aided Design*, 2004, pp.379-386.

[10] N. MacDonald, Broadcom Corp., *personal communication*, June 2013.

[11] T. G. Szymanski, "Computing Optimal Clock Schedules", *Proc. Design Automation Conference*, 1992, pp. 399-404.

[12] K. Wang and M. Marek-Sadowska, "Potential Slack Budgeting with Clock Skew Optimization", *Proc. International Conference on Computer Design*, 2004, pp. 265-271.

[13] C.-W. A. Tsao and C.-K. Koh, "UST/DME: A Clock Tree Router for General Skew Constraints", *ACM Transactions on Design Automation of Electronics System* 7(3) (2002), pp. 359-379.

[14] J. G. Xi and W. W.-M. Dai, "Jitter-Tolerant Clock Routing in Two-phase Synchronous Systems", *Proc. International Conference on Computer-Aided Design*, 1996, pp. 316-320.

[15] K. Wang, L. Duan and X. Cheng, "ExtensiveSlackBalance: An Approach to Make Front-end Tools Aware of Clock Skew Scheduling", *Proc. Design Automation Conference*, 2006, pp. 951-954.

[16] K. Wang, H. Fang, H. Xu and X. Cheng, "A Fast Incremental Clock Skew Scheduling Algorithm For Slack Optimization", *Proc. Asia and South Pacific Design Automation Conference*, 2008, pp. 492-497.

[17] X. Wei, Y. Cai and X. Hong, "Effective Acceleration of Iterative Slack Distribution Process", *Proc. International Symposium on Circuits and Systems*, 2007, pp. 1077-1080.

[18] "Cadence SOC Encounter User Guide". *http://www.cadence.com/products/di/first_encounter/pages/default.aspx*

[19] "OpenCores". *http://opencores.org*

[20] "Synopsys Design Compiler User Guide". *http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DCUltra/Pages/*

[21] "Synopsys IC Compiler User Guide". *http://www.synopsys.com/Tools/Implementation/PhysicalImplementation/Pages/*

[22] "Synopsys PrimeTime User's Manual". *http://www.synopsys.com/Tools/Implementation/Signoff/PrimeTime/Pages/*